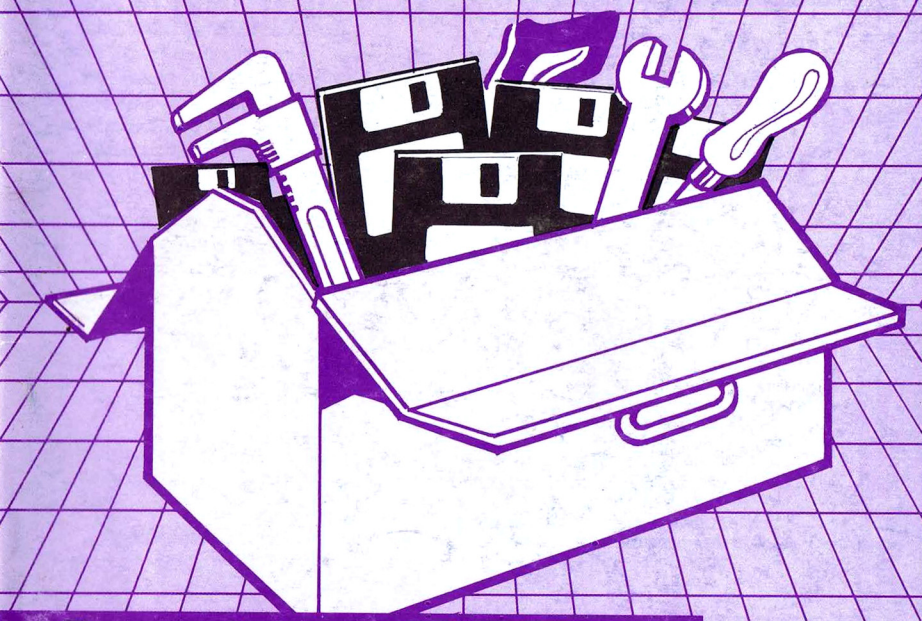
Volume 1
Issue 7

June
1988

Price £1.20

RISC USER



Risc User Toolbox

THE MAGAZINE AND SUPPORT GROUP
EXCLUSIVELY FOR USERS OF THE ARCHIMEDES

Volume 1 Issue 7
June 1988

RISC USER

CONTENTS

FEATURES

News	4
Beat Box	8
A Window on the Archimedes (Part 2)	15
Basic V - Array Parameters	19
Archimedes Visuals	20
Introducing ARM Assembler (Part 3)	22
3D Graphics (Part 3)	28
Hints & Tips	35

UTILITIES

Fancy Font Scroller	5
RISC User Toolbox	12

REVIEWS

Sigmasheet	10
SoundSynth	25
Watford Video Digitiser	26
Desk Top Stories	33
3D Graphics Development System	34

RISC User is published by BEEBUG Ltd.
Co-Editors: Mike Williams, Dr Lee Calcraft
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Production Assistant: Yolanda Turuelo
Technical Assistant: Lance Allison
Subscriptions: Mandy Mileham

BEEBUG, Dolphin Place, Holywell Hill, St.Albans, Herts AL1 1EX. Tel. St.Albans (0727) 40303

All rights reserved. No part of this publication may be reproduced without prior written permission of the Publisher. The Publisher cannot accept any responsibility whatsoever for errors in articles, programs, or advertisements published. The opinions expressed on the pages of this journal are those of the authors and do not necessarily represent those of the Publisher, BEEBUG Limited.

BEEBUG Ltd (c) 1988

Printed by Arlon House

The Archimedes Magazine
and Support Group.

EDITORIAL

As you will read in this month's news, Acorn has now officially announced price rises for most Archimedes systems and peripherals. These price rises derive principally from recent substantial increases in the prices of memory chips worldwide, and a number of other microcomputer manufacturers have already announced price increases for their machines. Following Acorn's reported loss of £3.3M last year no one could reasonably expect them to absorb this extra cost for more than a very short period.

However, there are some interesting features to note. The price of the entry level 305 series remains unchanged, thus increasing the price differential with respect to the 310 series. Maybe Acorn are hoping to boost sales of the 305 series as a result, but we would still recommend serious users to go for a full megabyte of memory with a 310.

The increase in the price of the 310 series has also been applied to the 0.5Mbyte upgrade that converts a 305 to a 310. In consequence, whereas most price increases are of the order of 9% or 10%, the increase on this upgrade is a hefty 67%. The I/O podule and MIDI add-on are also subject to small price increases.

As strong supporters of Acorn and the Archimedes we cannot but regret these increases, however justified they may be, and we can only hope that future sales are not dented as a result.

RISC USER MAGAZINE DISC

All the programs in this issue of the magazine plus:

1. An impressive demonstration of a new high resolution 256 colour mode from Computer Concepts (this requires a multi-sync monitor to appreciate this).

2. An extended version of the Beat Box program by Crosbie Fitch.

3. Sample screens captured with the aid of Watford Electronic's Digitiser, reviewed in this issue.

See the back cover for price and ordering details.

ACORN NEWS

Acorn has just held the first of a series of dealer meetings throughout the UK, the purpose of which is to keep dealers fully informed of the fast-moving Archimedes world. Harvey Coleman, Acorn's managing director, speaking at the meeting, said that Archimedes sales had meant a good first quarter for the company (profits approaching £1 million have been rumoured), and he went on to comment about the Archimedes winning the prestigious Tobie award from the electronics industry. Coleman was, however, quick to stress Acorn's continuing commitment to the Master series, with another 40,000 units now being produced.

Another question asked by many was why Acorn had called off the Acorn User show. There was no official comment from Acorn, although it has been rumoured that a new show, still run in conjunction with Acorn User magazine, will take place in October.

ACTING ON IMPULSE

Computer Concepts, in a surprise move, has revealed that they are developing an entirely new operating system for the Archimedes, to replace the much criticised Arthur. The new operating system, called 'IMPULSE', is being produced because Computer Concepts claim that the standard system is just not capable of supporting the advanced applications that they are working on. The new features in Impulse have not yet been finalised, but it is known that a major facility will be multi-tasking. Other features include extended windows and better graphics handling. When asked about this move, Charles Moir of Computer Concepts said "We are initially producing an operating system which loads from disc and takes over the Archimedes, although we are also working on a complete ROM-based replacement so that Arthur can be removed altogether. It is likely that we will release cut down versions of future products to run under Arthur, but the full-blown versions will need Impulse installed." It is not yet known exactly when Impulse will be available, but Computer Concepts promise a demonstration at the PCW show in September. Computer Concepts can be reached at Gaddesden Place, Hemel Hempstead, Herts HP2 6EX, or telephone (0442) 63933.

ARTHUR BITES BACK

Acorn have also been working on a new operating system for the past few months. Arthur 2, as the new system is called, is far more than just a bug-fixed version of the current Arthur 1.2. Acorn have not yet released any details of Arthur 2, but it is known that a major feature is the Desktop, now re-written in machine code, and supporting the long awaited multi-tasking. A complete RAM filing system is included, and the graphics primitives have been extended to include Bézier curve drawing; very useful for software designed to drive Postscript laser printers. The new operating system is already being supplied to software developers in the form of Arthur 1.7, although it is not known when it will be available to the general public. It seems almost certain that users will have to pay to upgrade to Arthur 2, but there are no details of the price yet.

ARCHIE PRICE INCREASE

Acorn have announced a price increase for the Archimedes, effective from 1st July. The rise brings the price of a 310 system with colour monitor to £1213 inc. VAT, and a colour based 440 will now cost you £3161 inc. VAT. Anybody who has ordered a 440 and is still waiting for delivery will unfortunately have to pay the extra. The reason for the rise is a worldwide increase in the cost of integrated circuits used in the Archimedes, with other manufacturers having to take similar action. BEEBUG will continue to supply machines at the old price while stocks last.

A PRESENTABLE PROGRAM

Lingenuity, the software division of Lindis International, has just released a presentation package for the Archimedes, called appropriately 'PRESENTER'. The system allows data that is entered manually or imported from a spreadsheet to be displayed as a bar, line or pie chart, and subsequentally dumped to both normal and colour printers. There is also a feature to let Presenter integrate with Artisan, Graphic Writer and 1st Word Plus. Details from: Lingenuity, Wood Farm, Linstead Magna, Halesworth, Suffolk IP19 0DU, or telephone (0986) 85476.

SOLID VISIONS

Silicon Vision, producers of the 3D graphics system reviewed in this issue, are working on a more advanced '3D Solids Design Animation System' for the Archimedes. The package should be available around the end of June, and users of the existing package will be able to upgrade for no more than the difference in price. RU

ALIAS FONT SCROLLER

by Barry Christie

Use this superbly smooth font scrolling routine to display messages of any kind. With little modification it could be used to display stylish program instructions, operate as a scrolling notice-board, an auto-cue, or for video titling.

Screen size must be set to 20 for this program to work.

The purpose of this program is to display scrolling text in Acorn's fancy fonts for messaging purposes. The size of font used is suitable for displays of all kinds, and the smoothness of scrolling allows the text to be read as it scrolls. The program is largely in Basic, and should prove easy to modify to suit individual needs.

The text to be scrolled is entered in DATA statements at the end of the program, as you can see from the accompanying listing. The program also responds to seven commands embedded within the text (see inset). These are all preceded by an asterisk, and should either be followed by a comma, or should appear as the last item in any DATA statement.

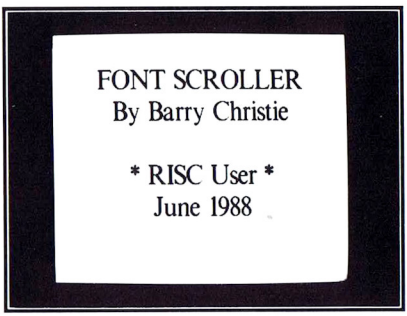
Embedded Commands

*L	Left justify all text
*R	Right justify all text
*C	Centre all text
*T,n	Tab by n graphics units
*S,n	Display n blank lines
*P,n	Pause for n centiseconds
*E	End the display

Where parameters are required, these should be given following a comma after the command. Thus the following sequence:

```
DATA *C
DATA A line of text
DATA *S,2
DATA *P,500
```

will display the phrase *A line of text* centred on the screen, and then pause for 5 seconds. Even though we have requested two blank lines to follow it (using *S,2), only one of these will appear on screen *before* the pause. This is because the display "lags" by one line. You will see how this works when you try the program, since it contains an extensive data section which generates a scrolling display of the commands and their effect. Note the use of quotation marks around text containing commas which are to be displayed.



When you type the program in, take particular care with the machine code section, and save the program before attempting to run it. The purpose of the machine code is to perform hardware scrolling. The program uses dual screens, and prints the display on the hidden screen before scrolling it into the screen on view. Although the program uses an 80K mode (mode 12), it therefore needs 160K of screen RAM. Set this with *CON.SCR.20 followed by Ctrl-Break. Before running the program you will also need a copy of Acorn's Trinity font from the Welcome disc. This should be installed (as the files IntMetrics and x90y45) on your current disc in the directory:

```
$.FONTS.TRINITY.MEDIUM.
```

You may if you wish alter the screen colours by adjusting lines 90 to 110. These three lines set up the border, foreground, and background colours respectively, by assigning their red, green and blue components.

```
10 REM              >FontasyC
20 REM Program      Alias Font Scroller
30 REM Version      A 0.8C
```

```
  40 REM Author    Barry Christie
  50 REM RISC User June 1988
  60 REM Program   Subject to Copyright
  70 :
  80 MODE 12:OFF
  90 VDU19,0,24,0,128,0
 100 rf=0    :gf=0    :bf=240
 110 rb=0    :gb=240  :bb=0
 120 ON ERROR MODE 0:REPORT:PRINT" at l
ine ";ERL:END
 130 PROCinitialise
 140 PROCfontscroll
 150 *FX112,0
 160 END
 170 :
 180 DEF PROCfontscroll
 190 REPEAT
 200 READ paint$
 210 IF paint$="" THEN paint$=" "
 220 CASE paint$ OF
 230 WHEN "*P"       : PROCfontdelay
 240 WHEN "*S"       : PROCfontspace
 250 WHEN "*L","*R"  : PROCfontalter
 260 WHEN "*C","*T"  : PROCfontalter
 270 WHEN "*E"       :
 280 OTHERWISE       : PROCfontpaint
 290 ENDCASE
 300 UNTIL paint$="*E"
 310 ENDPROC
 320 :
 330 DEF PROCfontdelay
 340 READ delay%
 350 TIME=0:REPEAT UNTIL TIME>delay%
 360 ENDPROC
 370 :
 380 DEF PROCfontspace
 390 READ totspace%:paint$=" "
 400 FOR fontspace%=1 TO totspace%
 410 PROCfontpaint
 420 NEXT
 430 ENDPROC
 440 :
 450 DEF PROCfontalter
 460 alter$=paint$
 470 fontL=alter$="*L"
 480 fontR=alter$="*R"
 490 fontC=alter$="*C"
 500 IF alter$="*T" THEN READ fontT ELS
E fontT=FALSE
 510 ENDPROC
 520 :
 530 DEF PROCfontpaint
 540 PROCpaintsetup
 550 PROCpaintpaint
 560 PROCpaintscrol
 570 ENDPROC
 580 :
 590 DEF PROCpaintsetup
 600 paintw%=-xgapsize%-1280
 610 FOR chr%=1 TO LEN(paint$)
 620 paintw%+=chrwidth%(ASC(MID$(paint$
,chr%,1)))
 630 NEXT
 640 paintx%=fontC*paintw%/2+fontR*pain
tw%+fontT
 650 painty%=1024-8*ygapsize%
 660 RECTANGLE FILL 0,painty%,1280,ygap
size%*4-1
 670 ENDPROC
 680 :
 690 DEF PROCpaintpaint
 700 FOR paint%=1 TO LEN(paint$)
 710 leter$=MID$(paint$,paint%,1)
 720 CALL scroller
 730 SYS "Font_Paint",0,leter$,&10,pain
tx%,painty%+ygapsize%
 740 paintx%+=chrwidth%(ASC(leter$))
 750 painty%+=4
 760 NEXT
 770 ENDPROC
 780 :
 790 DEF PROCpaintscrol
 800 FOR scrolling%=1 TO ygapsize%-LEN(
paint$)
 810 CALL scroller
 820 NEXT
 830 ENDPROC
 840 :
 850 DEF PROCinitialise
 860 *SET Font$Prefix $.FONTS
 870 PROCfontsetup(48,48)
 880 PROCfontchars
 890 PROCassemble
 900 paint$="*C":PROCfontalter
 910 ENDPROC
 920 :
 930 DEF PROCfontsetup(sizex%,sizey%)
 940 VDU 23,25,3,2,4,6,8,10,12,14
 950 VDU 23,25,136,9,rb,gb,bb,rf,gf,bf
 960 GCOL 128+8:CLG:GCOL 8:*FX112,2
 970 CLG
 980 fontx%=16*sizex%:fonty%=16*sizey%
```

```
  990 SYS "Font_FindFont",0,"Trinity.Med
ium",fontx%,fonty%,0,0 TO fonthandle%
 1000 SYS "Font_Paint",0,"A,a",&10,0,0
 1010 xgapsize%=0
 1020 ygapsize%=sizey%*2/3
 1030 ENDPROC
 1040 :
 1050 DEF PROCfontchars
 1060 DIM chrwidth%(255)
 1070 FOR chr%=33 TO 255
 1080 chrwidth%(chr%)=FNfontwidth(CHR$(c
hr%))+xgapsize%
 1090 NEXT
 1100 chrwidth%(32)=FNfontwidth(" A")-FN
fontwidth("A")
 1110 ENDPROC
 1120 :
 1130 DEF FNfontwidth(width$)
 1140 SYS "Font_StringBBox ",0,width$ TO
 x,x,x,width%,x
 1150 SYS "Font_ConverttoOS",0,width%,x
TO x,width%,x
 1160 =width%
 1170 :
 1180 DEF PROCassemble
 1190 DIM workarea 104:workarea!&00=150
 1200 workarea!&04=-1
 1210 SYS "OS_ReadVduVariables",workarea
,workarea
 1220 scrnconf%=!workarea
 1230 scrnsize%=&14000
 1240 basemodd%=scrnconf%
 1250 FOR pass=0 TO 2 STEP 2
 1260 P%=workarea
 1270 [ OPT pass
 1280 .scroller
 1290 LDR R0,baseaddr:LDR R1,basemodd
 1300 ADD R0,R0,#&140:CMP R0,R1
 1310 MOVEQ R0,#&00
 1320 STR R0,baseaddr:STR R0,swidata2
 1330 ADD R0,R0,#scrnsize%
 1340 CMP R0,R1:SUBGE R0,R0,R1
 1350 STR R0,swidata1
 1360 ADR R1,swiinfo1+&03
 1370 MOV R0,#&16:SWI "OS_Word"
 1380 MOV R0,#&13:SWI "OS_Byte"
 1390 ADR R1,swiinfo2+&03
 1400 MOV R0,#&16:SWI "OS_Word"
 1410 MOV R15,R14
 1420 .basemodd EQUD scrnconf%
 1430 .baseaddr EQUD &00000000
 1440 .swiinfo1:EQUD &01000000
 1450 .swidata1:EQUD &00000000
 1460 .swiinfo2:EQUD &02000000
 1470 .swidata2:EQUD &00000000
 1480 ]
 1490 NEXT pass
 1500 ENDPROC
 1510 :
 1520 DATA FONT SCROLLER
 1530 DATA By Barry Christie
 1540 DATA *S,4
 1550 DATA *P,200
 1560 DATA *L,TEXT - Left Justified
 1570 DATA Using the *L command
 1580 DATA *S,4
 1590 DATA *P,200
 1600 DATA *C,TEXT - Centred
 1610 DATA Using the *C command
 1620 DATA *S,4
 1630 DATA *P,200
 1640 DATA *R,TEXT - Right Justified
 1650 DATA Using the *R command
 1660 DATA *S,4
 1670 DATA *P,200
 1680 DATA *T,128,TEXT - TABbed
 1690 DATA *T,256,TEXT - TABbed
 1700 DATA *T,384,TEXT - TABbed
 1710 DATA *C,*S,1
 1720 DATA Tabbed, n=X coordinate
 1730 DATA "Using the *T,n command"
 1740 DATA *S,2,*P,200
 1750 DATA
 1760 DATA Spaces, n=no. of spaces
 1770 DATA "Using the *S,n command"
 1780 DATA *S,4
 1790 DATA *P,200
 1800 DATA Pause, n=seconds/100
 1810 DATA "Using the *P,n command"
 1820 DATA *S,4
 1830 DATA *P,200
 1840 DATA Specify end of text
 1850 DATA Using the *E command
 1860 DATA *S,4
 1870 DATA *P,200
 1880 DATA FONT SCROLLER
 1890 DATA By Barry Christie
 1900 DATA
 1910 DATA * RISC User *,June 1988
 1920 DATA *S,3,*P,400,*E,
```

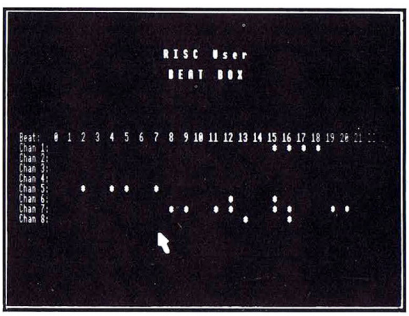
RU

BEAT BOX

by Crosbie Fitch

The author of the Welcome Music Editor presents a Rhythm Machine to create real-time rhythms on your Arc.

The Archimedes contains hardware capable of producing all manner of sounds and sound effects, all of which is supported at various levels in software. The accompanying program uses a range software calls to create a rhythm machine. This uses a nominal 24 beat bar, and allows the user to insert notes from any of the 8 active channels at any point within the bar, as the bar is cycled through in real time.



To make use of the program, type it in, and run it. You will see the bar set out before you, with the mouse pointer indicating the current position within the bar. This continually cycles around at a rate determined within the program. To insert a note from channel 1, press "1". You will hear it play, and see an asterisk entered at the current position on the bar. The note will now sound each time that the pointer passes this point. You may press any of the keys 1-8 at any point in the bar to create suitable rhythms. To erase any channel, just press Shift and a number key. Finally, note that the duration of each sound depends on how long you hold down the respective number key.

As you can see, the program presented here is relatively short, and is intended to provide a basis for experiment. As such, it lacks a number of features which might be found in a fully fledged rhythm machine. For example it has no provision for altering tempo. This may be accomplished manually (by adjusting lines 620 to 650), or by incorporating a special routine. Similarly, if you wish to set up a rhythm and then to play a melody, you will need to incorporate extra lines within the main loop (i.e. somewhere between lines 120-220) to send your melody to the appropriate sound queues while the rhythm continues. You may also like to experiment with this program using the Percussion relocatable module from the disc accompanying Issue 5.

This month's magazine disc contains an enhanced version of this program with the ability to save and load preset sequences, and to work with up to 12 different sequences, each selected at the press of a key. It also has variable tempo, and single-stepping to facilitate the entry of notes.

```
  10 REM             >BeatBox6
  20 REM Program     Drum Machine
  30 REM Version     A 0.6
  40 REM Author      Crosbie Fitch
  50 REM RISC User June 1988
  60 REM Program     Subject to Copyright
  70 :
  80 PROCinitialise
  90 ON ERROR PROCerror:END
 100 BEATS NBeats%:REM start beat count
 110 A%=0:T%=FALSE
 120 REPEAT
 130 B%=A%:A%=BEAT
 140 IF A%<B% PROCqueuerhythm
 150 IF A% EOR B% MOUSE TO B%*48+160,26
0
 160 SYS"OS_Byte",122 TO,K%
 170 IF T% THEN
 180 IF K%<>k% PROCdrumoff
 190 ELSE
 200 IF K%<52 C%=INSTR(K$,CHR$K%):IF C%
k%=K%:PROCdrumon
 210 ENDIF
 220 UNTIL FALSE
 230 :
 240 DEFPROCdrumon
 250 IF INKEY TRUE THEN
 260 M%=NOT(1<<C%-1)
 270 FOR I%=0 TO NBeats%-1
 280 R%?I%=R%?I% AND M%
```

```
  290 NEXT
  300 PRINTTAB(7,C%)SPC(NBeats%*@%)
  310 ELSE
  320 D%=BEAT:REM Start beat
  330 T%=TIME:REM Start time
  340 SOUND C%,V%,P%(C%),255
  350 VDU31,D%*@%+8,C%,46
  360 ENDIF
  370 ENDPROC
  380 :
  390 DEFPROCdrumoff
  400 R%?D%=R%?D% OR 1<<C%-1
  410 R%(C%)?D%=(TIME-T%) DIV 5
  420 T%=FALSE:SOUND C%,0,0,0
  430 VDU31,D%*@%+8,C%,42
  440 ENDPROC
  450 :
  460 DEFPROCqueuerhythm
  470 FOR I%=0 TO NBeats%-1
  480 IF R%?I% THEN
  490 F%=R%?I%<<1
  500 FOR J%=1 TO 8
  510 IF F% AND 1<<J% SOUND J%,V%,P%(J%)
,R%(J%)?I%,I%
  520 NEXT
  530 ENDIF
  540 NEXT:ENDPROC
  550 :
  560 DEFPROCinitialise
  570 VOICES 8:REM 8 channels
  580 FOR I%=1 TO 8
  590 SYS"Sound_AttachVoice",I%,I%+1
  600 NEXT
  610 K$="01"+CHR$17+CHR$18+CHR$19+CHR$2
4+"$"+CHR$21
  620 REM 24 beats per 2 second
  630 NBeats%=24:S%=2
  640 REM Tempo=&1000 x beats per csec
  650 TEMPO &1000*NBeats%/(100*S%)
  660 TIME=1:REM Reset timer
  670 V%=&17F:REM Max volume
  680 DIM R%(8):REM Duration per channel
  690 FOR C%=0 TO 8
  700 DIM R% NBeats%-1
  710 R%(C%)=R%
  720 NEXT
  730 R%=R%(0)
  740 FOR I%=0 TO NBeats%-1 STEP4
  750 R%!I%=FALSE
  760 NEXT
  770 DIMP%(8)
  780 REM set arbitrary pitches
  790 P%(1)=53:P%(2)=69:P%(3)=89:P%(4)=1
01
  800 :
  810 MODE 12:@%=3
  820 *POINTER
  830 COLOUR 0,0,48,144
  840 COLOUR 3
  850 PRINTTAB(32,6)"R I S C   U s e r"
  860 PRINTTAB(33,8)"B E A T   B O X"
  870 VDU28,1,31,79,14
  880 COLOUR 2
  890 PRINT"Beat: ";
  900 FOR I%=0 TO NBeats%-1
  910 PRINTI%;:NEXT:PRINT
  920 FOR C%=1 TO 8
  930 PRINT"Chan ";C%;":"
  940 NEXT:COLOUR 3:OFF
  950 ENDPROC
  960 :
  970 DEFPROCerror
  980 ON ERROR OFF
  990 MODE12
 1000 PRINT REPORT$;" at line ";ERL
 1010 VOICES 1:*CHANNELVOICE 1 1
 1020 ENDPROC
```

RU

Advertising in RISC User

RISC User has the largest circulation of any magazine dedicated to the Archimedes. In fact, we believe well over 50% of all Archimedes owners subscribe to the magazine, and many more read it. *RISC User* has to be the ideal advertising medium for all Archimedes products.

For more information and a rate card, contact:

Yolanda Turuelo
on (0727) 40303

SIGMASHEET

Reviewed by Mark Sealey

SigmaSheet is the latest spreadsheet published for the Archimedes, and the first written specifically for it (Logistix, reviewed in RISC User Issue 3, was originally released for the IBM and Atari ST ranges). The software is supplied on a 3.5" disc with accompanying manual.

SOME GENERAL COMMENTS

Whereas Logistix also has graphics, time management and other features not always associated with spreadsheets, SigmaSheet claims to be a fast, large-scale spreadsheet with no frills. However, SigmaSheet will exchange data with other Minerva software applications (such as System Delta and System Gamma), which greatly extends its capabilities (at a price). Moreover, SigmaSheet can import data from Lotus 123 as well as ViewSheet, Inter-sheet and the like. It will export ASCII files, and MS-DOS programs that use the CSV (Comma Separated Values) format are also catered for.

SigmaSheet has in fact a much greater capacity than Logistix, and claims to accommodate sheets greater than 16,000 rows by 1,000 columns. Such a huge size may be of little value to many potential users. However, it does make it possible to keep several separate sheets alongside one another, though as SigmaSheet has no windows option, you would have to scroll backwards and forwards, a nuisance even though scrolling is fast.

There are three further features to note before examining the software in more detail: SigmaSheet will make full use of a Floating Point Co-Processor if installed, but in any case will cope with numbers up to 17 decimal places. Screen modes supporting 132 columns are possible, as well as the more usual 80; with SigmaSheet's default of black on yellow, the 132 column display is surprisingly legible too. And the formats of individual cells can be set so that you could, if you wished, arrange for monthly negatives to be in a different colour to yearly ones.

USING SIGMASHEET

You enter SigmaSheet in its default state with 7 columns (width 9 characters) and 24 rows visible, and a standard column width of 9 characters. The screen is divided into two areas, the spreadsheet itself and various, continually updated displays - time of day, type of re-calculation, and the status and format of each cell etc. This is uncluttered and quite easy on the eye.

Cursor movement is smooth and logical, and controlled by the cursor keys, the Home key and the so-called GOTO function. All this has been well thought out. The Return key continues the direction in which you were already going, but this can be altered by means of the Options menu. It is possible to GOTO a label (e.g. "January" or "Rates") as well as a cell reference. What is more, unlike System Delta Plus, the Archimedes' function keys can all be programmed with strings or figures that are likely to be used frequently, thus speeding up data entry.

ENTERING DATA AND EDITING CELLS

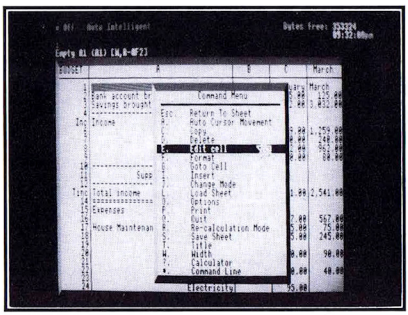
Most usual features of a spreadsheet are present: as well as normal data entry, columns, rows and whole blocks of cells can be replicated - both absolutely and relatively. Column widths can be changed, and a range of mathematical and Boolean operations performed on individual cells.

Data, labels, values and formulae are all detected intelligently on input. If you type a number, a value is assumed. If you enter a line of dashes to indicate a total at the foot of a column, SigmaSheet treats this as a label. It is useful not to have to preface every entry with its type. Editing is easy, and there is a considerable choice of display modes covering formulae, justification, and decimal places, as well as currency, date and the use of colours.

COMMANDS

The eighteen operations or commands include the use of star commands, and a numeric calculator. Each command may be accessed in one of three ways: by using the mouse menu button, by pressing the initial letter preceded by the now almost obligatory and ubiquitous backslash "/", or by using that same letter with the Control key.

Each command results in a pop-up menu. Several of these (the Print one, for instance) yield further menus as you proceed. Most give sensible default values so that just pressing Return often does the job. Whether you are put off by such a hierarchical menu system and prefer command driven packages, or whether you really can keep track of what is happening only you can judge.



What certainly is poor, however, is the way that the whole tree of menus can disappear on pressing Escape (if you decide not to continue with an option already selected) instead of just the most recent. Add to this the way that the remains of the previous menu are still visible behind the current one, and the fact that the boxes for some of the responses at these stages are not dynamic (i.e. you have a long row of empty squares), and some of it is more reminiscent of an early BBC B program. Rather clumsy, I'm afraid.

PERFORMANCE

There can be few complaints here. Scrolling is smooth, and the updating of cells is very fast. Indeed, the speeds achieved are extremely impressive: Minerva claim that recalculation of every cell in the whole 16,000 by 1,000 sheet with trigonometrical functions can be completed in just 24 seconds. Column insertion and width changes as well as replication are all but instantaneous.

SigmaSheet has one drawback in this respect though. When asked to indicate a range of cells (or even a single cell) it has become usual for spreadsheets to allow their selection by moving the cursor on the screen. Logistix also colours the appropriate cells as you go, for instance. With SigmaSheet, the only way is to enter the details of the cells in letters and numbers. This would be all very well (though not as user-friendly) if it were not that the menu controlling this was itself obscuring part of the sheet, making reference to it difficult.

Probably the most advanced feature of this spreadsheet is its provision of conditional maths functions. Not only are these very useful (although almost de rigeur on all serious spreadsheets), easy to use in this one, but also well explained in the manual: this can be a difficult area, but quite sophisticated results can be obtained comparatively easily.

CONCLUSIONS

The manual is good, much better than previous Minerva ones. It is clearly laid out with a good reference section and yet no index. There would also seem to be omissions - how to protect cells, for instance.

Compared with Logistix, with its time and task management facilities, its critical path analysis, graphics and database as well as inbuilt language, SigmaSheet appears to have less to offer But what it does, in the main it does very well. Don't be too put off by the drawbacks mentioned here - particularly if you intend to maintain compatibility with the other applications in Minerva's range. For anyone new to spreadsheets who doesn't need or is actively deterred by Logistix's multiplicity of features, SigmaSheet is a good serviceable product. And if you need immense capacity and speed (eight times that of Logistix), then you will certainly want to go for SigmaSheet. However, if ease of use, eye appeal and overall feel are of particular importance, I would suggest you consider paying the extra forty odd pounds and get the much more comprehensive Logistix.

RU

Product	**SigmaSheet**
Supplier	**Minerva Systems 69 Sidwell Street, Exeter, EX4 6PH.**
Price	**£69.95 (inc. VAT)**

RISC USER TOOLBOX

David Spencer kicks off a major new series that develops a comprehensive Toolbox module for the Archimedes.

Over the next few months we are going to build up a complete set of star commands that will be useful to programmers and other users. The whole 'toolbox' will be a relocatable module, which is simply extended when more commands are added. The design of the system is quite open-ended so that additional features can be easily added at a later date, and you can tailor the module to your own needs as you wish.

This first article describes a memory editor, which unlike that in RISC User Issue 3, is invoked by a star command, and can be used without having to load a program or corrupt any workspace. The routines that make up the editor will also be used in future issues to form the basis of other commands.

To set up the Toolbox module, including this month's memory editor, type in nand save the program listed. Make sure that the program is kept safely as this will form the shell for all future additions. Running the program will save the module to disc under the name TOOLBOX. To load in the module you should exit from Basic with QUIT, and then issue the command TOOLBOX.

The memory editor is then invoked by:

```
*MEDIT <address>
```

where <address> is the address, in hex, at which to start editing. The display will change to a mode 12 screen, and show the contents of memory in the form of the address, followed by sixteen bytes in hex, and the ASCII equivalent of those bytes. The cursor, which is indicated in reverse video, will be at the start address. The editor will not let you refer to any logical address that isn't actually mapped to RAM (see RISC User Issue 3).

Once in the editor, the cursor can be moved around with the cursor keys. Shift together with the cursor keys moves to the left or right of the current line, or a page up or down. The Tab key switches between hex and ASCII for data entry. When data is entered in ASCII format the cursor moves to the right after each character. The Delete key is equivalent to cursor left. To leave the editor press Escape.

Next month we will explain in more detail how the editor works, and build further on our Toolbox.

```
  10 REM >Toolsrc
  20 REM Program   Toolbox
  30 REM Author    David Spencer
  40 REM Version   A 1.0
  50 REM RISC User June 1988
  60 REM Program   subject to copyright
  70 :
  80 DIM code 10000
  90 FOR pass=0 TO 3 STEP 3
 100 P%=0:O%=code
 110 [OPT pass+4
 120 EQUD 0:EQUD init:EQUD 0
 130 EQUD 0:EQUD title:EQUD help
 140 EQUD tab:EQUD &C0040:EQUD swi
 150 EQUD switab:EQUD 0
 160 .title
 170 EQUS "RiscUserToolbox":EQUB 0
 180 .help
 190 EQUS "RU Toolbox":EQUB 9:EQUS "1.0
0 (22 May 1988)"
 200 EQUB 0:ALIGN
 210 .init:STMFD R13!,{R0-R3,R14}
 220 LDR R0,[R12]:CMP R0,#0
 230 LDMNEFD R13!,{R0-R3,PC}
 240 MOV R0,#6:MOV R3,#&400
 250 SWI "OS_Module":STR R2,[R12]
 260 LDMFD R13!,{R0-R3,PC}
 270 .tab
 280 EQUS "Medit":EQUB 0
 290 ALIGN:EQUD memedit
 300 EQUD &10001
 310 EQUD memsyn:EQUD memhlp
1320 EQUB 0
1330 .memhlp
1340 EQUS "*Medit invokes the scrolling
memory editor."
1350 EQUB 13
1360 .memsyn
1370 EQUS "Syntax: Medit <address>"
1380 EQUB 0:ALIGN
2390 .memedit
2400 STMFD R13!,{R14}:MOV R1,R0
2410 MOV R0,#16:SWI "OS_ReadUnsigned"
2420 MOV R0,R2:MOV R1,R2
2430 SWI "OS_ValidateAddress"
2440 LDMCSFD R13!,{PC}:SWI &100+22
2450 SWI &10C:STMFD R13!,{R0}
2460 MOV R1,R0
```

```
2470 .lowloop:SUB R2,R2,#16
2480 MOV R0,R2
2490 SWI "OS_ValidateAddress"
2500 BCC lowloop:ADD R2,R2,#16
2510 LDMFD R13!,{R0}:MOV R3,R0
2520 .hiloop
2530 ADD R3,R3,#16:MOV R1,R3
2540 SWI "OS_ValidateAddress"
2550 BCC hiloop:SUB R3,R3,#16
2560 MOV R1,#1:MOV R4,#0:MOV R5,#0
2570 .memedit2
2580 STMFD R13!,{R2-R5}:BL swi0
2590 MOV R6,R2:LDMFD R13!,{R2-R5}
2600 CMP R6,#27:ORRNE R1,R1,#1
2610 SWI &11F:SWI &100:SWI &11F
2620 BNE memedit2:LDMFD R13!,{PC}
2630 .switab
2640 EQUS "RUtools":EQUB 0
2650 EQUS "MemEdit":EQUB 0
3660 EQUB 0:ALIGN
3670 .swi
3680 LDR R12,[R12]
3690 CMP R11,#(swijumpend-swijump)/4
3700 ADDCC PC,PC,R11,LSL #2:B noswi
3710 .swijump:B swi0
4720 .swijumpend
4730 .noswi
4740 ADR R0,noswierr:CMP PC,#&80000000
4750 LDMFD R13!,{PC}
4760 .noswierr
4770 EQUD &1000:EQUS "Bad toolbox SWI"
4780 EQUB 0:ALIGN
4790 .curleft
4800 CMP R0,R2:MOVEQ PC,R14
4810 SUB R0,R0,#1:AND R5,R0,#15
4820 CMP R5,#15:BEQ curup2
4830 MOV PC,R14
4840 .curright
4850 ADD R5,R0,#1:CMP R5,R3
4860 MOVEQ PC,R14:MOV R0,R5
4870 AND R5,R0,#15:CMP R5,#0
4880 BEQ curdown2:MOV PC,R14
4890 .curdown
4900 ADD R0,R0,#16:CMP R0,R3
4910 BCC curdown2:SUB R0,R0,#16
4920 MOV PC,R14
4930 .curdown2
4940 STMFD R13!,{R0,R14}
4950 SWI &11F:SWI &100:SWI &11F
4960 ADD R0,R0,#15*16:CMP R0,R3
4970 BCC curdown3:SWI "OS_WriteS"
4980 EQUS STRING$(80," "):EQUB 0
4990 B curdown4
5000 .curdown3:BL prtlinex
5010 .curdown4:BL statline
5020 LDMFD R13!,{R0,PC}
5030 .curup CMP R0,#16:MOVCC PC,R14
5040 SUB R0,R0,#16:CMP R0,R2:BCS curup2
5050 .curup1:ADD R0,R0,#16:MOV PC,R14
5060 .curup2:STMFD R13!,{R0,R14}
5070 SWI &11E:SWI &10B
5080 SUBS R0,R0,#15*16:BMI curup25
5090 CMP R0,R2:BCS curup3
5100 .curup25:SWI "OS_WriteS"
5110 EQUS STRING$(79," "):EQUB 0
5120 B curup4
5130 .curup3:BL prtlinex
5140 .curup4:BL statline
5150 LDMFD R13!,{R0,PC}
5160 .byte0:STMFD R13!,{R2,R14}
5170 MOV R2,#0:SWI "OS_Byte"
5180 LDMFD R13!,{R2,PC}
5190 .prtlinex:STMFD R13!,{R0,R14}
5200 BIC R0,R0,#15:BL prtline
5210 LDMFD R13!,{R0,PC}
5220 .prtline:STMFD R13!,{R0-R5,R14}
5230 BIC R0,R0,#15:LDR R2,[R12,#12]
5240 ADD R0,R0,R2:CMP R1,#0
5250 ADD R1,R12,#32:MOV R2,#10
5260 BEQ twodig:SWI "OS_ConvertHex8"
5270 B addone
5280 .twodig:SWI "OS_WriteS"
5290 EQUS "    ":EQUB 0
5300 SWI "OS_ConvertHex4"
5310 .addone:SWI "OS_Write0"
5320 SWI "OS_WriteS":EQUS":  ":EQUB 0
5330 LDMFD R13,{R3}:MOV R4,#0
5340 .prtloop:ADD R1,R12,#32
5350 MOV R2,#10:LDRB R0,[R3,R4]
5360 SWI "OS_ConvertHex2"
5370 SWI "OS_Write0":SWI &120
5380 ADD R4,R4,#1:CMP R4,#16
5390 BNE prtloop:SWI &120:MOV R4,#0
5400 .prtloop2
5410 LDRB R0,[R3,R4]:CMP R0,#127
5420 MOVCS R0,#ASC".":CMP R0,#32
5430 MOVCC R0,#ASC".":SWI "OS_WriteC"
5440 ADD R4,R4,#1:CMP R4,#16
5450 BNE prtloop2:SWI "OS_WriteS"
5460 EQUS "   ":EQUB 0
5470 SWI "OS_NewLine"
5480 LDMFD R13!,{R0-R5,PC}
5490 .prtscr STMFD R13!,{R0-R4,R14}
5500 SWI &11E:BIC R0,R0,#15
5510 SUB R0,R0,#15*16:MOV R4,#31
5520 .prtscr2 CMP R0,#0:BLT prtscr25
5530 CMP R0,R2:BCS prtscr3
5540 .prtscr25:STMFD R13!,{R0}
```

```
5550 SWI &100+23:SWI &108:SWI &104
5560 SWI &106:MOV R0,#6:BL sndnull
5570 SWI "OS_NewLine":LDMFD R13!,{R0}
5580 B prtscr4
5590 .prtscr3:BL prtline
5600 .prtscr4 SUBS R4,R4,#1
5610 BEQ prtscr5:ADDS R0,R0,#16
5620 BMI prtscr2
5630 CMP R0,R3:BCC prtscr2:SWI &100+23
5640 SWI &108:SWI &104:SWI &10A
5650 MOV R0,#6:BL sndnull
5660 .prtscr5:BL statline
5670 LDMEQFD R13!,{R0-R4,PC}
5680 .remcur MOV R4,#1:B showcur1
5690 .showcur MOV R4,#0
5700 .showcur1 STMFD R13!,{R0-R4,R14}
5710 LDR R1,[R12]:MOV R1,R1,LSR #1
5720 SWI &11F:AND R0,R0,#15:CMP R1,#0
5730 ADDEQ R0,R0,R0,LSL #1
5740 ADDNE R0,R0,#49:ADD R0,R0,#11
5750 EOR R3,R1,#1:SWI "OS_WriteC"
5760 SWI &10F
5770 .showcur2 SWI &111:CMP R4,#0
5780 SWIEQ &180:SWINE &181
5790 MOV R0,#135:SWI "OS_Byte"
5800 SWI &111:CMP R4,#0:SWIEQ &181
5810 SWINE &180:MOV R0,R1
5820 SWI "OS_WriteC":SWI &111:SWI &180
5830 CMP R3,#0:MOVNE R3,#0
5840 BNE showcur2:LDMFD R13!,{R0-R4,PC}
5850 .sndnull:STMFD R13!,{R14}
5860 .sndnull2:SWI &100:SUBS R0,R0,#1
5870 BNE sndnull2:LDMFD R13!,{PC}
5880 .swi0:STMFD R13!,{R0-R6,R14}
5890 STR R4,[R12,#12]:STR R5,[R12,#16]
5900 AND R6,R1,#2:STR R6,[R12]
5910 CMP R0,R2:LDMCCFD R13!,{R0-R6,PC}
5920 CMP R0,R3:LDMCSFD R13!,{R0-R6,PC}
5930 MOV R6,R4:MOV R0,#237:MOV R1,#2
5940 BL byte0:STRB R1,[R12,#4]
5950 MOV R0,#225:MOV R1,#&A0
5960 BL byte0:STRB R1,[R12,#5]
5970 MOV R0,#229:MOV R1,#1
5980 BL byte0:STRB R1,[R12,#8]
5990 SWI &100+23:SWI &101:MOV R0,#8
6000 BL sndnull:LDMFD R13!,{R0-R3}
6010 BL prtscr:BL showcur
6020 .editloop:STMFD R13!,{R0}
6030 SWI "OS_ReadC":MOV R5,R0
6040 LDMFD R13!,{R0}:CMP R5,#27
6050 BEQ swi0_out:CMP R5,#9
6060 BEQ swap:CMP R5,#127
6070 BCC alter:BL remcur
6080 CMP R5,#127:MOVEQ R5,#&AC
6090 CMP R5,#&AC:BNE c1
6100 BL curleft:B keydone
6110 .c1 CMP R5,#&AD:BNE c2
6120 BL curright:B keydone
6130 .c2 CMP R5,#&AE
6140 BLEQ curdown:BEQ keydone
6150 CMP R5,#&AF:BLEQ curup
6160 BEQ keydone:CMP R5,#&8C
6170 BICEQ R0,R0,#15:BEQ keydone
6180 CMP R5,#&8D:ORREQ R0,R0,#15
6190 BEQ keydone:CMP R5,#&8E
6200 BNE notdn:ADD R5,R0,#16*31
6210 CMP R5,R3:MOVCC R0,R5
6220 BLCC prtscr:B keydone
6230 .notdn:CMP R5,#&8F:BNE swi0_out
6240 SUBS R5,R0,#16*31:BCC keydone
6250 CMP R5,R2:MOVCS R0,R5:BLCS prtscr
6260 .keydone BL showcur
6270 B editloop
6280 .alter:BL remcur
6290 LDR R4,[R12]:AND R4,R4,#2
6300 CMP R4,#0:BEQ hexalter
6310 STRB R5,[R0]
6320 .alter2:SWI &11F:SWI &100:SWI &10F
6330 BL prtlinex:LDR R4,[R12]
6340 AND R4,R4,#2:CMP R4,#0
6350 BLNE curright:BL showcur
6360 B editloop
6370 .hexalter:CMP R5,#ASC"0"
6380 BCC alter2:CMP R5,#ASC"G"
6390 BCS alter2:CMP R5,#ASC"A"
6400 SUBCS R5,R5,#7:CMP R5,#&40
6410 BCS alter2:AND R5,R5,#15
6420 LDRB R4,[R0]:ORR R4,R5,R4,LSL #4
6430 STRB R4,[R0]:B alter2
6440 .swap BL remcur:LDR R4,[R12]
6450 EOR R4,R4,#2:STR R4,[R12]
6460 B keydone
6470 .swi0_out
6480 STMFD R13!,{R0,R5}:SWI &100+23
6490 SWI &101:SWI &101:MOV R0,#7
6500 BL sndnull:MOV R0,#237
6510 LDRB R1,[R12,#4]:BL byte0
6520 MOV R0,#225:LDRB R1,[R12,#5]
6530 BL byte0:MOV R0,#229
6540 LDRB R1,[R12,#8]:BL byte0
6550 LDR R1,[R12]
6560 LDMFD R13!,{R0,R2,R4-R6,PC}
6570 .statline:STMFD R13!,{R14}
6580 SWI &11F:SWI &100:SWI &11F
6590 SWI "OS_WriteS"
6600 EQUS STRING$(79," "):EQUB 0
6610 LDMFD R13!,{PC}
6620 ]NEXT
6630 SYS "OS_File",10,"TOOLBOX",
&FFA,,code,O%
```

RU

A WINDOW ON THE ARCHIMEDES
(Part 3)

Felix Andrew concludes his articles on the WIMP Manager by explaining how windows are updated in response to user input, and shows how to add pop-up menus to your applications.

THE WIMP POLL ROUTINE

The heart of any WIMP program is the WIMP poll routine (PROCpoll). This makes a call to the WIMP manager to see if the user has made any changes to any of the windows, clicked any of the mouse buttons, or made a selection from any menu. Clearly, this routine must be executed repeatedly (line 130). The WIMP manager returns a number or *reason code* depending on the action just taken by the user. There are 10 different reason codes, and each has specific data which is returned in a parameter block (e.g. *close window* will tell us which window to close, whereas *mouse button clicked* would tell us where the mouse was, which button was pressed etc.). Because of the different types of information which the WIMP manager can return, a separate routine is needed to respond to each of the changes the user may make. The reason codes used in the program so far (see RISC User Issues 5 & 6) are described below.

Reason code 3: Close window request

The user has clicked on the close icon of a window. A **CloseWindow** call will shut any specified open window (using PROCclosewindow), but you may want to check that it is OK to shut this window first. The variable *handle* contains the handle of the window to close, so you can do any checking you want (e.g. a word processor might require you to save your work before closing the window).

Reason code 2: Open window request

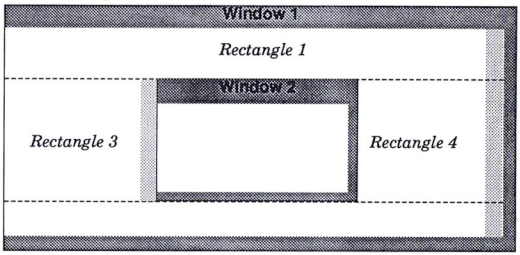
This code is returned after we try to open a window in PROCopenwindow1. A simple **OpenWindow** call will open the window for you (PROCopenwindow2).

Reason code 1: Redraw window request

The WIMP manager has had to redraw a window. This is the most complicated request, but is still quite easy to understand. The WIMP manager will try to do as much work as it can to keep the windows looking as they should, but it cannot keep track of what is in every window, and therefore what should be shown on the screen. It tells us which window needs attention, and which *rectangle* of the window needs to be redrawn. This way of using rectangles to indicate which area of the window needs to be redrawn is a very powerful one, but it can sometimes result in a very slow updating of windows. The bonus though is that it allows the user to place one window on top of another, while the window behind is scrolled.

THEORY

Let us suppose that you have the **Wave Form** window (window 3) underneath the **BEEBUG** window (window 2), as in Fig. 1, and that window 1 is scrolled to the right, i.e. the graphics are scrolled to the left. The WIMP manager first scrolls rectangle 1 to the left, and clears (to the WAE background colour) the shaded area at the right of rectangle 1. This is then repeated for rectangles 2, 3 and 4.



The easiest way you can see this happen is to display the windows as described (see last month), and then move the horizontal slider between the extremities of the scroll box by dragging the slider left and right. You can now see how the rectangles are filled. Obviously, as you scroll the window the WIMP manager has to scroll 4 rectangles and then fill them (when necessary). If you click in the scroll boxes you can see it's a bit slower, and much more flickery, than having to scroll just one area.

IMPLEMENTATION OF WINDOW CHANGES

The procedure PROCredrawwindow redraws the necessary windows. The variable *textcol* is set to the foreground colour of the WAE. The **ReDrawWindow** call causes the WIMP manager to redraw the window as best it can, and passes back the WAE co-ordinates of the first rectangle which it couldn't redraw. The procedure for drawing up that window's contents is then called depending on which window we are updating. Upon returning from the procedure we need to see if there are any more rectangles which the WIMP manager couldn't fill itself, and if necessary repeat the process. Once completed, the window should now contain all the graphics and text required.

POP-UP MENUS

Pop-up menus are the new feature we are adding to our demo program this month. Delete lines 120 and 125 from the program developed in parts 1 and 2, and then append the additional instructions listed at the end of this article (some of these lines replace existing ones). These changes will allow you to use a pop-up menu to open the windows that we were using previously (just click the *menu* button and *select* your choice). A second menu has also been implemented in the form of a *style* menu, obtained by clicking the menu button on the *style* label which appears on the screen when you run the complete program.

A pop-up menu is a form of window which consists of a title (maximum 11 chars), and a number of options in a column. The options are icons, which allows them to be both text and graphics, although for simplicity I have used only text.

Because the options can be of any size, you must specify the separation you want between each option. As I'm dealing with text only, this is set to 10*4 which allows room for a surrounding box. The menus can be as wide or as narrow as you like, but remember that they should be wide enough to contain the options you want displayed inside. You cannot change the separation of the options when calling the procedure PROCmenu, and the width is automatically calculated (line 2580).

Each option can have various flags associated with it. I will deal now with just three of these. The first one indicates that this option is the last on this menu. The others are much more useful. Any option can have a tick beside it, to show for example the default or current settings, or it can be shaded (non-selectable). This feature may be used to show the user options that might be available in other circumstances, or at other times. PROCmenu allows you to set the last two flags very easily by passing it a string comprising one character for each option, using a + to represent ticks, a - to denote non-selectable options, and a space if no flags are to be set.

CREATING YOUR OWN MENUS

The position of each menu, its colours, name, contents, flags and handle are all passed to the procedure PROCmenu as parameters. Although there are 12 of these in all, as before, they are quite easy to understand.

The first parameter, *menu*, is the menu handle. This is used in the same way as a window handle, but this time we specify its value. Next we specify where the top left-hand corner of the menu will appear on the screen (X,Y). If the co-ordinates given would cause any part of the menu to be drawn off the screen then it will be moved so that this cannot happen.

The foreground and background colours of the title and the title string are the next parameters to be passed (tf, tb, T$). These colours also specify the menu surround and title-bar background colours. Next we specify four different selection colours. The first parameter (pf) specifies the foreground colour of any text that the WIMP manager prints, while the next two parameters (pb, of) specify the text box background and foreground colours, and the foreground colour also determines the colour used for ticks and separation lines.

The last parameter has two functions. Firstly it specifies the background colour of the menu itself, and secondly it is used to change the colour of a text box when the pointer moves over it. This is done by exclusive ORing (EOR)

of its value with that of the text box foreground colour. It is a bit difficult to arrange the colours so that they look attractive when selected, but it's well worth trying.

Lastly we specify the text that will appear on the menu and what, if any, attributes each option has (O$,F$). Each option is separated from the next by a backslash (/). Two backslashes next to each other cause a section separator line to be drawn. This is a row of dashes which covers the full width of the menu. You cannot have two or more section separators following each other. The first option must not have a backslash before it, and the last option must be followed by a backslash. The attribute string is explained above, but by way of an example, if we had a menu with three items on it, and the first of them is to be non-selectable and the last one is to have a tick by it we would pass the string "- +". Looking at lines 6170 and 6230, which contains the calls to PROCmenu to produce the two menu displays, should help in understanding how to use PROCmenu.

ADDITIONAL REASON CODES

In order to make use of pop-up windows we need some additional reason codes, and these are briefly described below.

Reason code 9: Menu select

The user has made a choice from a particular menu. The procedure PROCmenuselect calls the procedure which deals with the particular menu from which the user has just made his selection. The variable *item1* specifies the option chosen by the user. The WIMP manager numbers the options starting from zero. The variable *menu* is the handle of the currently active menu.

Reason code 6: Mouse button change

The user has clicked with the mouse somewhere and may want the application to respond. This is dealt with by the procedure PROCmousebuttons. The WIMP manager returns the X and Y co-ordinates of the pointer. If the pointer was over a window then the handle of the window is returned (*handle*). If the pointer was also over an icon in that window, then the WIMP manager will return the handle of that icon (*iconh*). If the pointer wasn't over a window then *handle* is set to -1, and similarly *iconh* will be set to -1 if the pointer wasn't over an icon. It also returns the current state of the buttons (Z), and the state of the buttons on the previous button click (OZ).

Well, nobody said windows were easy. I suggest you digest the information above slowly, and with frequent reference to the program listing. That concludes for the time being our coverage of the WIMP manager, but we expect to return to this topic in the future.

```
  110 PROCopenwindow1(w4,0,1023,1280,40)
  250 w4=FNcreate(1280,-40,0,0,0,"",c0,c
1,0,0,pulldownf)
  255 dummy=FNaddicon(w4,100,0,100,-36,t
box,c2,0,"Style",string)
  265 M3F$="+ - "
  490 DIM block% 512,mbox% 9,menu% 1024
  520 REM Icon flags
  530 ictext=1:icsprite=2:icborder=4
  540 txtch=8:txtcv=16:icback=32
  550 ichelp=128:icind=256:txtrj=512
  560 icnonsel=2^22
  640 REM 'Macro Icon' flags
  650 tbox=ictext+txtch+txtcv
  660 sbox=icsprite:string=-1:sprite=-2
  760  WHEN 6:PROCmousebuttons
  770  WHEN 9:PROCmenuselect
  780  OTHERWISE
  950 :
  960 DEF PROCmousebuttons
  970 X=offset!0:Y=offset!4:Z=offset!8
  980 handle=offset!12:iconh=offset!16
  990 OZ=offset!20:CZ=Z EOR OZ
 1000 CASE handle OF
 1010  WHEN -1:PROCclickondesk
 1020  WHEN w4:PROCpulldowns
 1030 ENDCASE
 1040 ENDPROC
 1050 :
 1060 DEF PROCmenuselect
 1070 item1=offset!0
 1080 CASE menu OF
 1090  WHEN 1:PROCdeskmenu
 1100  WHEN 3:PROCstylemenu
 1110 ENDCASE
 1120 ENDPROC
 1780 :
 1790 DEF FNaddicon(HA%,X%,Y%,W%,H%,F%,F
```

```
C%,BC%,T$,T%)
 1800 $block%=STRING$(36,CHR$0)
 1810 block%!00=HA%:block%!04=X%
 1820 block%!08=Y%+H%:block%!12=X%+W%
 1830 block%!16=Y%
 1840 block%!20=(F%+(FC%<<24)+(BC%<<28))
 1850 IF T%=string OR T%=sprite THEN
 1860  T$=LEFT$(T$,11)
 1870  $(block%+24)=T$:VDU7
 1880 ELSE
 1890 numchars=EVAL(T$)
 1900 block%!20=block%!20 OR 2^8 OR (15<
<12)
 1910 block%!24=T%:block%!28=-1
 1920 block%!32=numchars
 1930 ENDIF
 1940 SYS "Wimp_CreateIcon",,block% TO h
andle
 1950 =handle
 1960 :
 2280 DEF PROCmenu(Menu,X,Y,tf,tb,T$,pf,
pb,of,ob,O$,F$)
 2290 LOCAL X%,N%,width,height,no,tick
 2300 T$=LEFT$(T$,11)
 2310 $(menu%+00)=T$
 2320 menu%?12 =tf:menu%?13 =tb
 2330 menu%?14 =pf:menu%?15 =pb
 2340 menu%!20 =10*4:menu%!24 =1*4
 2350 N%=1:O%=0:W%=LEN T$
 2360 WHILE N%<=LEN O$
 2370  X%=INSTR(O$,"/",N%)
 2380  op$=MID$(O$,N%,X%-N%)
 2390  op$=LEFT$(op$,11)
 2400  IF LEN op$=0 THEN
 2410  !(menu%+28+(O%-1)*24+0)=2
 2420  ELSE
 2430  no=0:tick=0
 2440  IF MID$(F$,O%+1,1)="-" THEN no=2^
22
 2450  IF MID$(F$,O%+1,1)="+" THEN tick=
1
 2460  IF LEN op$>W% THEN W%=LEN op$
 2470  !(menu%+28+O%*24+0)=tick
 2480  !(menu%+28+O%*24+4)=-1
 2490  !(menu%+28+O%*24+8)=((of<<24)+(ob
<<28)+%1101) OR no
 2500  REM Menu Icon Flags p.451
 2510  $(menu%+28+O%*24+12)=op$
 2520  O%+=1
 2530  ENDIF
 2540  N%=X%+1
 2550 ENDWHILE
 2560 O%=O%-1
 2570 !(menu%+28+O%*24+0)=(!(menu%+28+O%
*24+0)) OR &80
 2580 menu%!16=(W%+2)*16
 2590 SYS "Wimp_CreateMenu",,menu%,X,Y
 2600 menu=Menu
 2610 ENDPROC
 2620 :
 2630 DEF FNsetmenu(M$,O%,F$)
 2640 =LEFT$(M$,O%-1)+F$+MID$(M$,O%+1)
 2650 :
 2660 DEF FNmenustatus(M$,O%)
 2670 =MID$(M$,O%,1)
 2680 :
 2690 DEF FNmenutick(M$)
 2700 =INSTR(M$,"+")
 6000 DEF PROCdeskmenu
 6010 CASE item1 OF
 6020  WHEN 0:PROCopenwindow1(w1,100,400
,560,150)
 6030  WHEN 1:PROCopenwindow1(w2,1000,90
0,270,100)
 6040  WHEN 2:PROCopenwindow1(w3,200,900
,750,1000)
 6050  WHEN 3:PRINTTAB(0,0)"Shut up":qui
t=TRUE
 6060 ENDCASE
 6070 ENDPROC
 6080 :
 6090 DEF PROCstylemenu
 6100 IF item1=-1 THEN GOTO 6130
 6110 M3F$=FNsetmenu(M3F$,FNmenutick(M3F
$)," ")
 6120 M3F$=FNsetmenu(M3F$,item1+1,"+")
 6130 ENDPROC
 6140 :
 6150 DEF PROCpulldowns
 6160 CASE iconh OF
 6170  WHEN 0:PROCmenu(3,40,1023-36,c0,c
1,"Style",c1,c1,c2,c3,"Bold/Italic/Under
line/Outline/",M3F$)
 6180 ENDCASE
 6190 ENDPROC
 6200 :
 6210 DEF PROCclickondesk
 6220 IF Z AND 2^1 THEN
 6230 PROCmenu(1,X-6*16,Y,c0,c1,"Main Me
nu",c1,c2,c1,c3,"RISC User/BEEBUG/Wave F
orm//Quit/","     ")
 6240 ENDIF
```

RU

Basic V

ARRAY PARAMETERS

This month Mike Williams continues the discussion on parameter passing with a look at arrays.

One of the major limitations in the use of arrays in BBC Basic in the past has been the fact that they could not be used as parameters when calling functions and procedures. Fortunately, that has all changed with the advent of Basic V on the Archimedes, but there is a little more to it than first meets the eye.

Here is a simple example of a procedure to list the contents of an array:

```
1000 DEF PROClistarray(A())
1010 LOCAL i,n:n=DIM(A(),1)
1020 FOR i=1 TO n
1030 PRINT A(i)
1040 NEXT i
1050 ENDPROC
```

The name of the array, A, is specified as a parameter with empty brackets to signify that it is an array and not just a simple variable. The procedure declares two local variables (i and n), and n is set equal to the size of the array. This in itself is a very useful technique, and uses the DIM statement to return a value, in this case the size of the first dimension of the array (hence the 1). If the array has more than one dimension, then by changing the 1 to 2 or whatever, you can find the sizes of the other dimensions. By this means, a procedure or function can always determine for itself the size of an array passed to it as a parameter.

Each element of the array in turn is then printed. In the past, such a procedure could only be written by allowing the array A to be treated globally, i.e. referring to it both from within and from without the procedure as A. But this then has the disadvantage that the procedure will only print out the contents of that array and none other.

The beauty of using an array as a parameter is that the same procedure may be used with many different arrays, for example:

```
PROClistarray(ages())
PROClistarray(totals())
```

And by using the DIM statement to determine the size of an array inside the procedure, the arrays don't even have to be of the same size.

Last month we saw how variables passed as parameters could return values as well using the keyword RETURN. However, when arrays are used as parameters this happens automatically, and no RETURN is required. Thus, to sort the contents of an array we could write:

```
1000 DEF PRCsortarray(A())
1010 LOCAL i,j,n:n=DIM(A(),1)
1020 FOR i = n-1 TO 1 STEP -1
1030 FOR j = 1 TO i
1040 IF A(j)>A(j+1) SWAP A(j),A(j+1)
1050 NEXT:NEXT
1060 ENDPROC
```

Such a procedure, for example, could be called with:

```
PROCsortarray(numbers())
```

and the contents of the array numbers(), assumed to be dimensioned in the calling program, will be re-ordered by this procedure.

Unlike the use of single value variables, whenever an array is specified in the calling of a procedure or function, it is effectively a pointer to the start of the array in memory which is passed, and not the array itself. Acorn claims that actually to copy all the values from an array into a local array within the procedure (equivalent to what happens with ordinary variables) would take too long and result in large areas of memory being rapidly eaten up by data storage. In effect every array would be duplicated each time it was passed as a parameter. In practice, Acorn has probably chosen the right route, but it does make the logic behind the use of RETURN in ordinary parameter passing somewhat inconsistent.

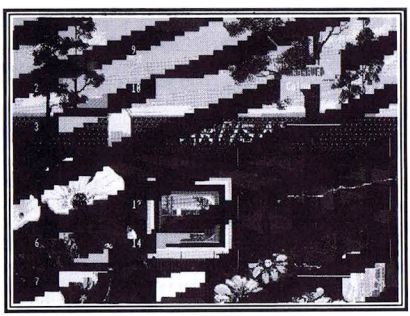
You must also remember, that as a result of the way arrays are passed as parameters, that there is no way in which you can avoid the array contents being changed if they are changed within the procedure definition. However, that is a very small price to pay for a powerful and highly desirable feature. **RU**

More smart routines to exploit the Arc's excellent graphics capabilities.

Dual Screen Cross-fader

By Nick van Someren

This program will cross-fade between any two 80K screens of the same mode (e.g. mode 12 or mode 13), leaving both intact after the process. In fact it generates a whole family of cross-fade patterns - from hosts of vertical or horizontal bars to a variety of dotted effects, and even a straight horizontal sweep. To run the program you will need 160K of screen RAM (*CON.SCR.20 then Ctrl-Break). You also need two suitable screens. The first (Screen1) should be a straight *ScreenSave image. The second should be *SAVEd (use for example the Fast-Save routine from Issue 3).



When you run the program it will request an input. Give any number between 1 and 5119 which does not have either 2 or 5 as a factor. The two screens will then be cross-faded with a pattern which depends on the number input. Pressing Return on its own repeats the last fade. Here are some numbers to try:

```
    1 5119   (l-r & r-l sweep)
   11   27     119  477  1121
 1713 1799    1911 1901  2221
```

Requires screensize 20

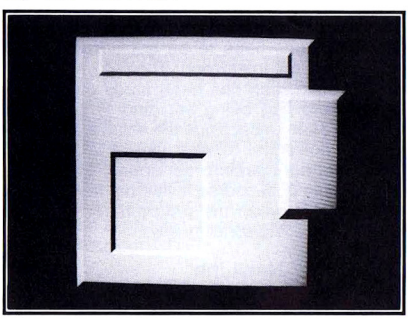
Listing 1

```
 10 REM              >CrossFade6
 20 REM Program      Screen Cross-Fade
 30 REM Version      A 0.6
 40 REM Author       Nick Van Someren
 50 REM RISC User June 1988
 60 REM Program   Subject to Copyright
 70 :
 80 DIM code% 1000
 90 FOR pass%=0 TO 3 STEP 3
100 P%=code%
110 [OPT pass%
120 .start
130 ;R0 Source scrn;R1 Dest Scrn
140 ;R2 Screen len;R3 Step;R4 delay
150 ;R8 X posn;R9 Y posn
160 STMFD R13!,{R14}
170 MOV R12,R4:MOV R4,#0
180 .loop
190 BL delay:AND R9,R4,#&3F
200 MOV R8,R4,LSR#6:BL blit
210 ADD R4,R4,R3:CMP R4,R2
220 SUBCS R4,R4,R2
230 BNE loop
240 LDMFD R13!,{PC}
250 .blit
260 ;Blit 4 bytes by 4 lines
270 ;320 bytes / line
280 ;80 blocks along by 64 high
290 STMFD R13!,{R2,R3,R10,R11}
300 ADD R10,R9,R9,LSL#2
310 ADD R10,R8,R10,LSL#6
320 ADD R11,R1,R10,LSL#2
330 ADD R10,R0,R10,LSL#2
340 LDR R2,[R10]:LDR R3,[R11]
350 STR R3,[R10],#320
360 STR R2,[R11],#320:LDR R2,[R10]
370 LDR R3,[R11]:STR R3,[R10],#320
380 STR R2,[R11],#320:LDR R2,[R10]
390 LDR R3,[R11]:STR R3,[R10],#320
400 STR R2,[R11],#320:LDR R2,[R10]
410 LDR R3,[R11]:STR R3,[R10],#320
420 STR R2,[R11],#320
430 LDMFD R13!,{R2,R3,R10,R11}
440 MOV PC,R14
450 .delay
460 STMFD R13!,{R11}:MOV R11,R12
470 .delayloop
480 SUBS R11,R11,#1
490 MOVNV R0,R0:MOVNV R0,R0
500 MOVNV R0,R0:MOVNV R0,R0
510 MOVNV R0,R0:MOVNV R0,R0
520 BNE delayloop
530 LDMFD R13!,{R11}:MOV PC,R14
540 ]
550 NEXT
```

```
560 MODE 12
570 *SCREENLOAD Screen1
580 *LOAD Screen2 1FEC000
590 A%=&1FEC000:C%=80*64
600 B%=&1FD8000:E%=63
610 Q%=477
620 REPEAT:REPEAT:REPEAT
630 VDU 30:ON
640 INPUT D%:OFF
650 IF D%=0 D%=Q%
660 UNTIL D%>0 AND D%<C%
670 Q%=D%:F%=D%:G%=C%
680 REPEAT
690 G%=G% MOD F%:SWAP G%,F%
700 UNTIL F%=0:UNTIL G%=1
710 CALL start
720 UNTIL FALSE
```

Generalised Plinth Procedure

By Lee Calcraft



This program presents a more flexible version of the plinth procedure which appeared in Issue 3. It has been generalised to produce plinths of any aspect ratio, and these can now be "sunken" as well as raised. The bulk of the listing provides examples of the way in which the procedure is called. It has 11 parameters. The first five give graphics dimensions: the X and Y co-ordinates of the bottom left hand corner of the plinth, its width and height, and the width of the 3D edging. The second five give the numbers of the colours used for the following parts of the display: bright streak, light border, face, dark border and dark streak respectively. The two "streaks" appear at the top left and bottom right of the plinth, and may not be visible in the screen-shot. Finally, the eleventh parameter indicates whether the plinth is raised or sunken. Both raised and "sunken" plinths assume a light source at the top left hand corner of the screen.

Listing 2

```
  10 REM            >Plinth2
  20 REM Program    Generalised Plinth
  30 REM Version    A 0.5
  40 REM Author     Lee Calcraft
  50 REM RISC User  June 1988
  60 REM Program    Subject to Copyright
  70 :
  80 MODE12
  90 COLOUR1,240,176,0
 100 COLOUR2,224,144,0
 110 COLOUR3,144,80,0
 120 COLOUR4,128,80,0
 130 COLOUR8,176,224,224
 140 COLOUR9,144,192,192
 150 COLOUR10,112,160,160
 160 COLOUR11,0,96,96
 170 COLOUR12,0,112,80
 180 :
 190 PROCplinth(0,0,1000,1000,30,8,9,10
,11,12,TRUE)
 200 PROCplinth(100,850,800,100,16,7,1,
2,3,4,TRUE)
 210 PROCplinth(850,300,280,500,40,8,9,
10,11,12,TRUE)
 220 PROCplinth(150,150,400,400,20,8,9,
10,11,12,FALSE)
 230 END
 240 :
 250 DEFPROCplinth(X,Y,WX,WY,W,C0,C1,C2
,C3,C4,raised)
 260 IF NOT raised SWAP C1,C3:SWAP C0,C
4
 270 GCOLC1:RECTANGLE FILL X,Y,WX,WY
 280 GCOLC3:RECTANGLE FILL X+W,Y,WX-2*W
,W
 290 RECTANGLE FILL X+WX-W,Y,W,WY-W
 300 MOVE X,Y:MOVE X+W,Y:PLOT85,X+W,Y+W
 310 MOVE X+WX-W,Y+WY-W:MOVE X+WX,Y+WY-
W
 320 PLOT85,X+WX,Y+WY:PLOT85,X+WX,Y+WY
 330 GCOLC0:LINE X,Y+WY,X+W,Y+WY-W
 340 GCOLC4:LINE X+WX,Y,X+WX-W,Y+W
 350 GCOLC2:RECTANGLE FILL X+W,Y+W,WX-2
*W,WY-2*W
 360 ENDPROC
```

RU

INTRODUCING ARM ASSEMBLER (3)

by Lee Calcraft

This Month: Using SWI Calls

The Archimedes' resident modules, including the Arthur operating system itself, contain many routines of potential use to the machine code programmer, and Acorn have documented a considerable number of these in the *Programmer's Reference Manual*. Each may be called from ARM assembler using the SWI instruction, or from Basic with the SYS command. In this article we will take a look at some of the simpler SWI calls, and in particular those concerned with output to the screen.

OS_WriteC	Write character in R0
OS_NewLine	Send CR LF sequence
OS_Write	Write embedded character
OS_WriteS	Write following string (0-term)
OS_Write0	Write any string (0-term)
OS_WriteN	Write any fixed length string

Selected SWI Calls

SWI OS_WriteC

This routine sends to the currently selected output stream (usually the screen) the character whose ASCII code is held in register R0. With one notable exception (OS_WriteI - see later) all SWI calls may be referenced by name or number. By way of illustration, both of the following sequences will send ASCII"Z" to the screen:

```
MOV R0,#ASC("Z")
SWI "OS_WriteC"
```

or

```
MOV R0,#ASC("Z")
SWI 0
```

Of these two forms the second is obviously the more concise but the least memorable, and Acorn advise the use of the full textual form of the call. We will also follow this convention, but you should be warned that the SWI string must always be given in the exact form in which it appears in the manuals, and must always appear within quotation marks *regardless* of what it says in the manuals. If you get so much as the case of a single letter wrong the SWI will not be recognised. The bar which follows the letters "OS" in each call is, incidentally, a *shifted* minus.

If you want to test out the SWI calls as we discuss them, you are referred to listing 1. This demonstrates the use of each in sequence.

CONDITIONAL SWIS

A further feature of OS_WriteC, shared by almost all SWI calls (OS_WriteS is the exception - see later) is that it can be used in conjunction with any of the condition mnemonics discussed last month. Thus for example, if you try the following sequence:

```
MOVS R0,ASC("Z")
SWIEQ "OS_WriteC"
```

nothing will happen! This is because we have added the "S" suffix (also discussed last month) to the initial MOV instruction, causing the processor's flags to be set according to the result of the instruction. The SWI which follows this instruction has been made conditional on the zero flag being set, and of course it will have been unset since the value loaded into R0 was *not* equal to zero (ASCII"Z" is 90). The SWI is not therefore performed. This particular example is of course trivial, but there are many instances when the programmer will need to perform SWIs conditionally.

SWI OS_NewLine

This very simple SWI performs a Carriage Return - Line Feed sequence. Its behaviour is independent of the contents of any of the ARM's user registers at the time of calling, and all registers are left undisturbed after the call. See listing 1 for examples of its use.

SWI OS_WriteI

This rather unusual SWI is the only one which cannot be called by name. It is called by number, and the number of the call also contains the value of the byte which the SWI outputs. The base number for this SWI is &100, and it is used as follows:

```
SWI &100+ASC("A")
```

This causes ASCII "A" to be output. You will see that unlike OS_WriteC, the user does not

need to load R0 with the value to be sent: it is already encoded directly in the instruction itself. In fact OS_WriteI is actually a whole family of SWIs whose SWI numbers run from &100 to &1FF.

SWI OS_WriteS

SWI OS_WriteS is one of three SWIs discussed here which may be used for sending strings of characters to the VDU. It achieves this by making repeated calls to OS_WriteC to send each character in turn. The string sent by OS_WriteS must be situated immediately following the SWI instruction itself, and must be terminated with a zero byte, as the following example illustrates:

```
SWI "OS_WriteS"
EQUS "Message"
EQUB 0
```

When this sequence of code is executed, the word "Message" will be displayed. As you can see, we have used two assembler directives to install the text string. EQUS is used for the string itself, while EQUB 0 places a zero byte as terminator at the end of the string. As you can see from listing 1 we have used the ALIGN directive to align the following instruction to a word boundary. This is not strictly necessary since the SWI normally copes with alignment, but may be considered a wise precaution.

When using OS_WriteS there is one important proviso: this SWI must *not* be made conditional. The reason for this is that if the SWI is *not* executed because the applied condition fails, the program will crash because it unexpectedly encounters a string of text where it was expecting a new instruction.

OS_Write0

In the previous case the string to be displayed was stored immediately following the corresponding SWI. With OS_Write0 and OS_WriteN the text string is located elsewhere. It is pointed to by the contents of R0, and need not even be located at a word boundary. This approach has two advantages. It means that sequences of machine code are not broken up by text strings, and that because of this, these two SWIs may be performed conditionally, since the corresponding text strings will no longer confuse the processor. One further point to note about SWI OS_Write0 is that the final character of the SWI name is a *zero* and not an upper case "O".

Listing 1 contains an example of the use of SWI OS_Write0, and you will note that we have used **ADR R0,text** to load R0 with a pointer to the text string. We have not encountered this form before. Oddly enough **ADR** is not a new ARM instruction, but an *assembler directive* whose function is to ensure that the named register is loaded with the required address pointer in an address independent manner. It does this by referring to the address of the pointer in terms relative to the program counter (R15) rather than in absolute terms.

OS_WriteN

This SWI is similar to the above except that output does not stop when a zero byte is encountered, as may be seen from the example in listing 1. The great advantage of this is that zero bytes may therefore be included as part of the string to be sent, making it suitable for sending VDU commands and so on. Instead of indicating the end of the sequence with a zero terminator, OS_WriteN uses the length of the string, as supplied in register R1. For example:

```
ADR R0,text
MOV R1,#7
SWI "OS_WriteN"
:
; Other instructions
:
.text
EQUS "Message"
```

Here register R0 is loaded with a pointer to the text string, and R1 with the number 7, indicating the number of characters to be displayed.

In listing 1, two examples of the use of this SWI are given. The first sends a string of text uninterrupted by an embedded zero byte. The second redefines character number 255 to generate an empty box shape. It uses the assembler equivalent of the VDU23 command:

```
VDU23,255,0,255,129,129,129,129,255,0
```

Again the string contains zero bytes, but this is no longer an impediment. Finally, OS_WriteN has the further advantage that it uses the low-level VDU drivers directly, and is much more efficient than the other two string routines discussed here, which both make repeated calls to OS_WriteC. On the negative side, this means that the call is *not* suitable for use when output streams other than the VDU are involved.

OS_Byte

Our brief look at ARM assembler SWIs would not be complete without a mention of OS_Byte. This SWI enables the machine code programmer to execute FX calls, and the Archimedes equivalent of the BBC Micro's OSBYTE calls. In ARM assembler this is just a case of setting up register R0 with the OS_Byte number, and optionally R1 and R2 with additional data. As an example, I have used FX219. This redefines the Tab key to generate any other keyboard character. In Basic the command *FX219,65 will cause the Tab key to generate the letter "A" (ASCII 65) whenever it is pressed.

To achieve the same in ARM assembler, the following sequence should be used:

```
MOV R0, #219
MOV R1, #ASC("A")
MOV R2, #0
```

After running the program in listing 1, you will see that pressing the Tab key now generates an "A". For details of other OS_Byte calls, and other SWI calls in general, you are referred to the *Programmer's Reference Manual*; though we will of necessity be returning to SWI calls at a later date.

Listing 1

```
  10 REM >3-1ARMpg6
  20 REM SWI Demos
  30 REM by Lee Calcraft
  40 :
  50 DIM space &1000
  60 FOR pass=0 TO 1
  70 P%=space
  80 [
  90 OPT pass*3
 100 .start              ; OS_WriteC
 110 MOV R0,#ASC("Z")
 120 SWI "OS_WriteC"
 130 :
 140 MOVS R0,#ASC("A")
 150 SWIEQ "OS_WriteC"
 160 ;                   OS_NewLine
 170 SWI "OS_NewLine"
 180 ;                   OS_WriteI
 190 SWI &100+ASC("I")
 200 :
 210 SWI "OS_NewLine"
 220 ;                   OS_WriteS
 230 SWI "OS_WriteS"
 240 EQUS "Demonstration of OS_WriteS"
 250 EQUW &0D0A
 260 EQUB 0
 270 ALIGN
 280 ;                   OS_Write0
 290 ADR R0,text
 300 SWI "OS_Write0"
 310 SWI "OS_NewLine"
 320 ;                   OS_WriteN
 330 ADR R0,text
 340 MOV R1,#48
 350 SWI "OS_WriteN"
 360 SWI "OS_NewLine"
 370 ;                   OS_WriteN for
 380 ADR R0,chardef    ; character defn
 390 MOV R1,#14
 400 SWI "OS_WriteN"
 410 :
 420 MOV R0,#219
 430 MOV R1,#ASC("A")
 440 MOV R2,#0
 450 SWI "OS_Byte"
 460 :
 470 MOV PC,R14        ; Return
 480 :
 490 .text
 500 EQUS "String terminated with VDU0"
 510 EQUB 0
 520 EQUS " plus some more text"
 530 .chardef
 540 EQUB 23         :EQUB 255
 550 EQUD &8181FF00 :EQUD &00FF8181
 560 EQUB 255        :EQUW &0A0D
 570 ]
 580 NEXT
 590 CALL start
```

Next month we will look at stacks, subroutines and memory access in general. RU

SOUNDSYNTH REVIEW

by Mark Sealey

Soundsynth is the first in an ambitious series of Archimedes music programs, collectively called Arpeggio, produced by Electromusic Research (EMR). EMR are well known for their music systems, and it was they who developed the MIDI (Musical Instrument Digital Interface) for the Archimedes.

When the disc is first booted you are presented with the main Arpeggio menu, offering 18 different packages in the whole suite, although Soundsynth is the only option available at the present time. Soundsynth has a quite specific job, namely to produce, manipulate and store waveforms (akin to the ENVELOPE parameters from former BBC machines) using EMR's own Wave Filing System (or WFS) to store the sounds produced. These sounds can be used by other programs, for example to change the instruments used by other music packages, such as Acorn's Music Editor on the Archimedes Welcome disc; or to provide sounds for your own programs; or for use in other parts of the Arpeggio system.

It is important to realise, therefore, that Soundsynth is not a complete synthesiser. For example, there are no facilities to allow sequencing or varied playback of the waveforms that are produced. So what is its interest? Is it not too specialised to be of any real value? Not at all. Firstly, because Soundsynth is interactive in nature, you can see the changes made to the sounds represented in clear graphs and thus learn a lot about the way sound behaves in this environment. A second reason for buying and using Soundsynth is its ease of use. To have such a multiplicity of variables and parameters as those that go with controlled sound output is potentially very confusing. In fact, although the Soundsynth display uses 132 columns and has the usual aircraft cockpit-like appearance of music programs, the screens are not at all confusing, and optional on-screen help is available at all times.

Each sound produced by Soundsynth can be made up from as many as 512 separate waveforms, and these are shown graphically on the screen as the sound is being edited. There are basic commands to LOAD and SAVE sounds on disc and also facilities for such functions as copying, moving, inserting, overlaying and deleting sections of waveforms, and also for editing each waveform graphically. To start editing a sound, for example, the mouse pointer is positioned on the graph and the wave you want simply 'shaped' with the mouse before fixing it as part of the sound, or listening to it on the Archimedes' own speaker. It is regrettable that there is no 'undo' function, so you need to think twice before making any major changes. Soundsynth also offers advanced editing features, which not only allow alteration of the sound by mathematical equation, but also echo, reversal, looping, slapback and gate on or off (a means of controlling amplitude in relation to the sound trigger).

A nice feature of Soundsynth is the ability to edit real sounds sampled using the Armadillo sound sampler reviewed in RISC User Issue 5.

Soundsynth is supplied with a twenty page manual. While this does cover everything, certain areas, particularly those concerning advanced features, are barely adequate. Once again, we have a good product let down by the standard of the documentation.

In my view, Soundsynth provides very good value for money, my only niggle being the manual, as mentioned earlier. The ability to edit waveforms on the screen is particularly impressive, as is the fact that waveforms can be printed in a number of different ways. Other packages to be offered in the Arpeggio system include a Performer, a Notator and MIDI Bank suites. These are all eagerly awaited.

RU

Product	**Soundsynth**
Supplier	**Electromusic Research 14 Mount Close, Wickford, Essex. Tel. (0702) 335747**
Price	**£39.95 inc. VAT (Introductory Offer Price)**

WATFORD'S VIDEO DIGITISER

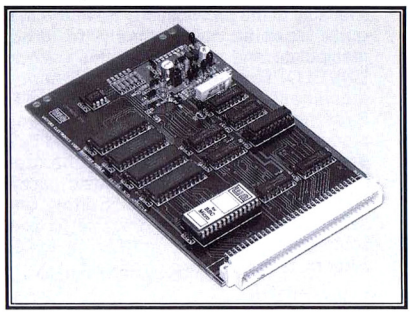
Lee Calcraft reviews Watford Electronics' Video Digitiser for the Archimedes

Watford Electronics' new video digitiser is supplied as a half width podule, and comes with manual and support disc. The podule is easily installed in a podule backplane socket (an optional extra with the 300 series). When the computer is switched on after installation, it gives a power-up message to the effect that the digitiser software is installed, and you will discover that the amount of free RAM in your machine has dropped by some 24K. This is because the digitiser board contains a relocatable module in EPROM. At power-up, this is downloaded into the machine's RMA area ready for use. You can prevent this happening by software-unplugging the module (with *UNPLUG Digitiser).

At the rear of the podule there is a single BNC socket, and this will accept video signals from most sources (1 volt nominal). For the purposes of this review, I used a National Panasonic video recorder (in playback mode), and a Ferguson Videostar colour camera. Both functioned well with the podule, which may also be used with monochrome sources. The sparsely populated podule board contains four 32K RAM chips, and these are used for directly "grabbing" incoming video images. The stored image can then be transferred to the Archimedes screen in a variety of modes. This permits a number of functions to be carried out before the image reaches the screen, such as scaling, rotating, smoothing, and so on.

In spite of the considerable tasks required of the podule and accompanying software, the speed of screen grabbing and display is excellent. A mode 9 screen for example, can be displayed in real time at the rate of 12.5 frames per second for a full screen, or 25 frames per second as a quarter-size screen. The worst time is for a full-screen mode 20 display, which runs at between 5 and 7 frames per second.

Regardless of the resolution of the currently selected Archimedes screen, the digitiser always operates with a 512x256 image stored internally with 64 grey levels. How this is represented on the screen is largely up to the user, though it should be made clear that the system is essentially a mono one. No colour information is stored; though a software option does allow the user to make thee successive grabs using red, green and blue filters in front of the camera. No filters are however supplied with the unit, and the composite image produced by this technique requires manual colour adjustment.

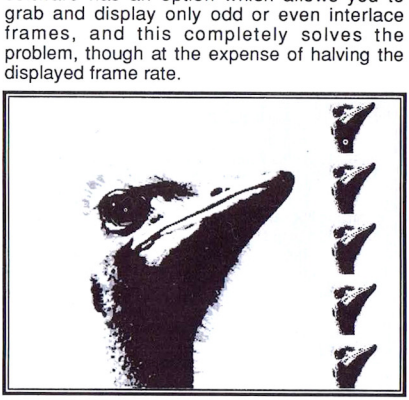


The Watford Digitiser

The quality of the mono image captured by the system is very good, though of course it depends on the quality of the originating video image from camera or video recorder. Even more critical is the effect of attempting to display the 512x256 pixel image with its 64 grey levels on an Archimedes screen. Whatever mode you choose, compromises must be made. In terms of representing the full grey scale, you get the best results by using mode 15 on a black and white mono monitor. The software uses the 64 colours to recreate the 64 levels. In all other modes, the 64 grey levels are reduced to 16 (or less), with resultant picture degradation.

As far as the pixel resolution is concerned, modes 12 and 15 can both cope with the full resolution of the grabbed picture. But mode 20 (multi-sync monitors only) permits a higher vertical resolution, since the software allows the odd and even interlace scans of successive fields to be integrated into a high resolution picture (though of course with only 16 grey

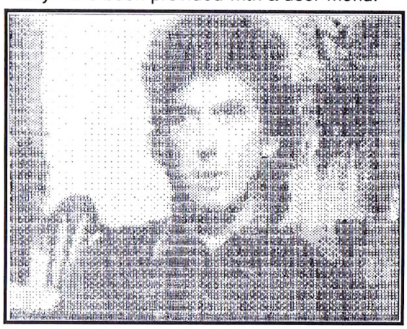
levels). In fact in all modes but 20 the alternate even and odd interlace scans generated by a video camera or recorder can cause jitter on certain detailed images. Consequently, the software has an option which allows you to grab and display only odd or even interlace frames, and this completely solves the problem, though at the expense of halving the displayed frame rate.



High resolution screen

In designing the software, Mike Harrison who also designed the hardware and wrote the manual, has done a first rate job. He has provided some 20 SWI calls for frame grabbing, image processing and display, and these are well documented in the manual. There are also a number of useful star commands which make it very easy to control the unit, either directly from the keyboard, or via simple programs in Basic. For example *Grab will grab and display a frame in the current mode. *See will continuously grab and display frames until a key is pressed, while *ShowPic will display the picture currently in the digitiser's memory. An interesting command, *MakeSprite, allows the creation of sprites from a captured image. By using the mouse, the image is instantly and continuously scaled in both X and Y directions at breathtaking speed. When you are happy with the exact size and aspect ratio of the image, you can save this as a sprite for use in any program. There are also star commands for fast saving and loading both screens, and video images. A number of printer dumps further complement the package.

Finally, the support disc contains a collection of example programs which show off the use of some of the module's many commands. One interesting program grabs a sequence of frames, and then replays them as an animated sequence. Another, called "Telly", displays a constantly updated video image, and is useful for setting up the system - though I would have found it more useful in the early stages if it had trapped the "No video signal" error. Each of the demonstration programs is documented with REM statements, and gives useful ideas on using the many SWI calls provided by the system. However, I would have appreciated a little more about each program in the manual itself, and these programs should really have been provided with a user-menu.



Using the supplied printer dump

But these are only minor gripes. The manual, the podule and module software are all first class and a credit to their author. As to price, I cannot square the podule's relatively low chip count with the £249 plus VAT price tag without thinking of the costs of running a Rolls Royce. But if you want one you will Jessa have to pay up. Maybe, as Mike Harrison commented, the software alone is worth the price.

This month's magazine disc contains some screens captured using the digitiser. **RU**

Product	**Real-Time Video Digitiser**
Supplier	**Watford Electronics** **250 High Street,** **Watford, WD1 2AN.** **Tel. (0923) 37774**
Price	**£286.35 inc. VAT**

3D GRAPHICS

(Part 3)

In the third and final part of his series on 3D graphics, Mark Davis explains how to shade solid objects on the screen.

So far we have covered the drawing of simple wire-frame objects in Basic, and shown how to draw objects in ARM assembler, with hidden-line removal. The next step after this is to colour-fill the surfaces to make a solid which appears illuminated from a particular direction, and that is what we will look at this month.

The drawing of solid objects is really an extension of hidden line elimination, except that the surfaces are filled with a solid colour, rather than just drawn as outlines. This month's listing is therefore an adaptation of the one given last time. Running the new program displays the house-shaped object as before, but now, the sides, base and roof are in different colours. As with the previous programs, the house rotates automatically until the space bar is pressed, and then rotation is controlled by the mouse. The *select* and *menu* buttons enlarge and reduce the shape.

The program uses a 256 colour mode (mode 13), because of its larger choice of colours, and different shades of the same colour can then be used to give shadow effects. By altering the shade of the colour, depending on the angle from which the surface is viewed, and how far away it is, gives a much more realistic effect. The shade is calculated according to the angle of the face with respect to the light source, and how far away it is. The shape is defined so that it appears to be illuminated from the front.

There are three main differences between the program here and last month's. Lines 1610-1940 are responsible for drawing the faces of the shape, and because these are now solid they are drawn as a series of triangles. Lines 2120-2260 are responsible for the shading effects. This routine takes a colour number in R2 and reduces its brightness by the value in R0. Finally, the data structure used to store the shape contains an extra parameter for each face. This is a number between 0 and 255 that defines the colour of the surface at full brightness, and these are added to the end of the set of data statements.

Another feature incorporated in the revised program is 'bank switching'. This concept uses two screens in memory, so that while one is being updated, the other is displayed. When the update is finished the screens can be switched over and the process repeated. The result is that you don't actually see the shape being redrawn. Bank switching is explained in the *User Guide* and the *Programmer's Reference Manual*.

This now concludes our brief survey of the subject of 3D graphics for the time being. If you want to experiment further you can either use the programs given here as a template, or incorporate the techniques into your own programs.

```
 10 REM >SolidDraw
 20 REM Program   3D Drawing
 30 REM Version   A 1.0
 40 REM Author    Mark Davis
 50 REM Risc User May 1988
 60 REM Program   Subject to copyright
100 DIM code &10000
110 coords=9:surfs=9
120 MODE141:ORIGIN 640,512:PROCass:CLS
130 bank%=1:xrot=0:yrot=-90:zrot=90
140 zpos=100:PROCinit:PROCdemo
150 *POINTER
160 MOUSE RECTANGLE-640,-512,1280,1024
170 MOUSE TO yrot,xrot
180 REPEAT SYS 6,112,bank%
190 bank%=bank% EOR 3:SYS 6,113,bank%
200 WAIT:CLS:MOUSE yrot,xrot,B
210 IF B AND4 THEN zpos+=4*(zpos>32)
220 IF B AND2 THEN zpos+=4
230 PROCrotate:IF I%>0 THEN250
240 !zoff=zpos*256:CALL draw
250 UNTIL FALSE
260 END
270 :
280 DEFPROCinit:RESTORE
290 FOR X=0 TO coords-1:READ x,y,z
300 !(xw+X*4)=x*256:!(yw+X*4)=y*256
310 !(zw+X*4)=z*256:NEXT
320 FOR X=0 TO surfs-1:FOR Y=0 TO 3
330 READ surf:?(surfdat+X*4+Y)=surf
340 NEXT:NEXT
350 FOR X=0 TO surfs-1:READ X?coldat
360 NEXT
370 ENDPROC
380 DATA 10,-10,-10,  10,10,-10
390 DATA 10,10,10,    10,-10,10
400 DATA -10,-10,-10,-10,10,-10
```

```
410 DATA -10,10,10,  -10,-10,10
420 DATA 18,0,0
430 DATA 1,5,6,2,2,6,7,3,5,4,7,6
440 DATA 0,4,5,1,0,3,7,4,0,1,8,8
450 DATA 1,2,8,8,2,3,8,8,3,0,8,8
460 DATA 191,127,136,127
470 DATA 191,195,195,195,195
480 :
490 DEFPROCrotate
500 A%=(xrot+720) MOD 360
510 B%=(yrot+720) MOD 360
520 C%=(zrot+720) MOD 360
530 I%=USR rotate
540 ENDPROC
550 :
560 DEFPROCass:PRINT"Please wait..."
570 FOR PASS=0 TO 2 STEP 2
580 P%=code
590 [OPT PASS
600 .xa EQUD0:.ya EQUD0:.za EQUD0
610 .xro EQUD0:.yro EQUD0:.zro EQUD0
620 .xoff EQUD 0
630 .yoff EQUD 0
640 .zoff EQUD 80*256
650 .rotate STMFD R13!,{R14}
660 MOV R12,#0:ADR R8,xw:ADR R9,sin
670 ADD R10,R9,#360:ADR R14,xr
680 MOV R0,R0,ASL#2:MOV R1,R1,ASL#2
690 MOV R2,R2,ASL#2:STR R0,xro
700 STR R1,yro:STR R2,zro
710 .rotlp
720 LDR R0,xro:LDR R1,yro:LDR R2,zro
730 LDR R3,[R8]:LDR R4,[R8,#yw-xw]
740 LDR R5,[R8,#zw-xw]
750 LDR R6,[R9,R2]:LDR R7,[R10,R2]
760 MUL R11,R3,R7:MOV R2,R11,ASR#8
770 MUL R11,R4,R6:SUB R2,R2,R11,ASR#8
780 STR R2,xa:MUL R11,R4,R7
790 MOV R2,R11,ASR#8
800 MUL R11,R3,R6:ADD R2,R2,R11,ASR#8
810 STR R2,ya:LDR R6,[R9,R1]
820 LDR R7,[R10,R1]:LDR R3,xa
830 LDR R4,ya:MUL R11,R3,R7
840 MOV R2,R11,ASR#8:MUL R11,R5,R6
850 SUB R2,R2,R11,ASR#8:STR R2,xa
860 MUL R11,R5,R7:MOV R2,R11,ASR#8
870 MUL R11,R3,R6:ADD R2,R2,R11,ASR#8
880 STR R2,za:LDR R6,[R9,R0]
890 LDR R7,[R10,R0]:LDR R3,xa
900 LDR R5,za:MUL R11,R4,R7
910 MOV R2,R11,ASR#8:MUL R11,R5,R6
920 SUB R2,R2,R11,ASR#8:STR R2,ya
930 MUL R11,R5,R7:MOV R2,R11,ASR#8
940 MUL R11,R4,R6:ADD R2,R2,R11,ASR#8
950 STR R2,za:LDR R3,xa:LDR R0,xoff
960 ADD R3,R3,R0:STR R3,[R14]
970 LDR R4,ya:LDR R0,yoff
980 ADD R4,R4,R0:STR R4,[R14,#yr-xr]
990 LDR R5,za:LDR R0,zoff
1000 ADD R5,R5,R0:STR R5,[R14,#zr-xr]
1010 CMP R5,#0:MOVLT R0,#1
1020 LDMLTFD R13!,{R15}:MOV R7,R14
1030 BL recaddr:MOV R14,R7
1040 BIC R5,R5,#&30
1050 LDR R7,[R6,R5,LSR#4]:ADR R6,xs
1060 MUL R11,R3,R7:MOV R11,R11,ASR#6
1070 STR R11,[R6,R12,ASL#2]!
1080 MUL R11,R4,R7:MOV R11,R11,ASR#6
1090 STR R11,[R6,#ys-xs]
1100 ADD R8,R8,#4:ADD R14,R14,#4
1110 ADD R12,R12,#1:CMP R12,#coords
1120 BCC rotlp:MOV R0,#0
1130 LDMFD R13!,{R15}
1140 .hiddcheck
1150 STMFD R13!,{R14}
1160 ADR R4,surfdat:ADD R4,R4,R0,ASL#2
1170 LDRB R1,[R4]:LDRB R2,[R4,#1]
1180 LDRB R3,[R4,#2]:ADR R5,xs
1190 LDR R6,[R5,R1,ASL#2]
1200 MOV R6,R6,ASR#2
1210 LDR R7,[R5,R2,ASL#2]
1220 MOV R7,R7,ASR#2
1230 LDR R8,[R5,R3,ASL#2]
1240 MOV R8,R8,ASR#2:ADR R5,ys
1250 LDR R9,[R5,R1,ASL#2]
1260 MOV R9,R9,ASR#2
1270 LDR R10,[R5,R2,ASL#2]
1280 MOV R10,R10,ASR#2
1290 LDR R11,[R5,R3,ASL#2]
1300 MOV R11,R11,ASR#2
1310 SUB R0,R7,R6:SUB R1,R11,R9
1320 MUL R2,R0,R1:MOV R2,R2,ASR#8
1330 SUB R0,R8,R6:SUB R1,R10,R9
1340 MUL R3,R0,R1
1350 SUB R0,R2,R3,ASR#8
1360 LDMFD R13!,{R15}
1370 .xw EQUS STRING$(coords*4,CHR$0)
1380 .yw EQUS STRING$(coords*4,CHR$0)
1390 .zw EQUS STRING$(coords*4,CHR$0)
1400 .xr EQUS STRING$(coords*4,CHR$0)
1410 .yr EQUS STRING$(coords*4,CHR$0)
1420 .zr EQUS STRING$(coords*4,CHR$0)
1430 .xs EQUS STRING$(coords*4,CHR$0)
1440 .ys EQUS STRING$(coords*4,CHR$0)
1450 .surfdat
1460 EQUS STRING$(surfs*4,CHR$0)
1470 .coldat
1480 EQUS STRING$(surfs,CHR$0):ALIGN
1490 .sin EQUS STRING$(250,CHR$0)
1500 EQUS STRING$(110,CHR$0)
1510 .cos EQUS STRING$(250,CHR$0)
1520 EQUS STRING$(250,CHR$0)
1530 EQUS STRING$(250,CHR$0)
1540 EQUS STRING$(250,CHR$0)
```

```
1550 EQUS STRING$(220,CHR$0)
1560 EQUS STRING$(220,CHR$0)
1570 .xsa EQUD xs
1580 .ysa EQUD ys
1590 .surfdata EQUD surfdat
1600 .coldata EQUD coldat
1610 .plot STMFD R13!,{R14}
1620 MOV R12,R0:BL hiddcheck:CMP R0,#0
1630 LDMGEFD R13!,{R15}
1640 BL facecol:LDR R1,coldata
1650 LDRB R2,[R1,R12]:BL minus
1660 SWI &112:SWI &100:AND R0,R2,#&3F
1670 SWI 0:SWI &117:SWI &111:SWI &102
1680 AND R0,R2,#&C0:SWI 0:SWI &100
1690 SWI &100:SWI &100:SWI &100
1700 SWI &100:SWI &100:MOV R9,R12
1710 LDR R0,surfdata
1720 ADD R12,R0,R9,ASL#2:LDR R10,xsa
1730 LDR R11,ysa
1740 LDRB R0,[R12,#0]
1750 LDR R1,[R10,R0,ASL#2]
1760 MOV R1,R1,ASR#8
1770 LDR R2,[R11,R0,ASL#2]
1780 MOV R2,R2,ASR#8:MOV R0,#4:SWI &45
1790 LDRB R0,[R12,#1]
1800 LDR R1,[R10,R0,ASL#2]
1810 MOV R1,R1,ASR#8
1820 LDR R2,[R11,R0,ASL#2]
1830 MOV R2,R2,ASR#8:MOV R0,#4:SWI &45
1840 LDRB R0,[R12,#3]
1850 LDR R1,[R10,R0,ASL#2]
1860 MOV R1,R1,ASR#8
1870 LDR R2,[R11,R0,ASL#2]
1880 MOV R2,R2,ASR#8:MOV R0,#85:SWI &45
1890 LDRB R0,[R12,#2]
1900 LDR R1,[R10,R0,ASL#2]
1910 MOV R1,R1,ASR#8
1920 LDR R2,[R11,R0,ASL#2]
1930 MOV R2,R2,ASR#8:MOV R0,#85:SWI &45
1940 LDMFD R13!,{R15}
1950 .face EQUD0
1960 .faceno EQUD surfs
1970 .draw STMFD R13!,{R14}
1980 MOV R0,#0:STR R0,face
1990 .drwlp BL plot:LDR R0,face
2000 ADD R0,R0,#1:STR R0,face
2010 LDR R1,faceno:CMP R0,R1:BCC drwlp
2020 LDMFD R13!,{R15}
2030 .facecol STMFD R13!,{R14}
2040 ADR R2,fcols:MOV R3,#0:MOV R4,#0
2050 RSB R0,R0,#0
2060 .facllp
2070 LDR R5,[R2,R3,ASL#2]:CMP R0,R5
2080 MOVCC R4,R3:ADD R3,R3,#1
2090 CMP R3,#8:BCC facllp
2100 MOV R0,R4:LDMFD R13!,{R15}
2110 .minus STMFD R13!,{R14}
2120 .milp
2130 SUBS R0,R0,#1:LDMMIFD R13!,{R15}
2140 CMP R2,#64:SUBCS R2,R2,#64
2150 BCS milp:TST R2,#&3:SUBNE R2,R2,#1
2160 TST R2,#&C:SUBNE R2,R2,#4
2170 TST R2,#&30:SUBNE R2,R2,#16
2180 ORR R2,R2,#&C0
2190 B milp
2200 .fcols EQUD &40000
2210 EQUD&40000*11/12:EQUD&40000*10/12
2220 EQUD &40000*9/12:EQUD &40000*8/12
2230 EQUD &40000*7/12:EQUD &40000*6/12
2240 EQUD &40000*5/12:EQUD &40000*4/12
2250 EQUD &40000*3/12:EQUD &40000*2/12
2260 EQUD &40000*1/12
2270 .recaddr ADR R6,recip:MOV R15,R14
2280 .recip
2290 ]
2300 NEXT
2310 FOR X%=0 TO 449:I%=X%*4:I%!sin=SIN
(RAD(X%))*&100:NEXT
2320 FOR X%=1 TO 12799:I%=recip+X%*4:!I
%=200000/X%:NEXT
2330 ENDPROC
2340 :
2350 DEFPROCdemo:!zoff=25600
2360 FOR xrot=0 TO -20 STEP -2
2370 PROCrotate
2380 SYS 6,112,bank%:bank%=bank% EOR 3
2390 SYS 6,113,bank%
2400 WAIT:CLS:CALL draw
2410 NEXT
2420 FOR yrot=-90 TO 90 STEP 4
2430 PROCrotate
2440 SYS 6,112,bank%:bank%=bank% EOR 3
2450 SYS 6,113,bank%
2460 WAIT:CLS:CALL draw:NEXT
2470 I=INKEY(0):IF I=32 ENDPROC
2480 FOR zrot=90 TO 360 STEP 6
2490 PROCrotate
2500 SYS 6,112,bank%:bank%=bank% EOR 3
2510 SYS 6,113,bank%
2520 WAIT:CLS:CALL draw:NEXT
2530 I=INKEY(0):IF I=32 ENDPROC
2540 FOR X=0 TO 180 STEP 4
2550 zrot=X+4:yrot=96+X:xrot=-24-X
2560 PROCrotate
2570 SYS 6,112,bank%:bank%=bank% EOR 3
2580 SYS 6,113,bank%
2590 WAIT:CLS:CALL draw:NEXT
2600 REPEAT:yrot+=4:PROCrotate
2610 SYS 6,112,bank%:bank%=bank% EOR 3
2620 SYS 6,113,bank%
2630 WAIT:CLS:CALL draw
2640 UNTIL INKEY(-99)
2650 yrot=yrot MOD 360
2660 ENDPROC
```

RU

3D CAD/ANIMATION SYSTEM

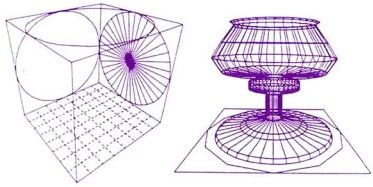
ARCHIMEDES SOFTWARE

Once again Silicon Vision steals the lead with this incredible offer. It's true we're offering our international, best-selling 3D Graphics Development System (3D GDS) at an incredibly low price. No we haven't gone mad, it's simply our way of introducing you to the world of 3D computer graphics.

3D GDS is a full blown 3D CAD & Animation system that can handle 3D models of any complexity. The package consists of a single disc providing the Design & Animation facilities and a library of predefined models to get you started. A comprehensive 95 page manual describes all the facilities including tutorials and example 3D animation programs on disc. The standalone 3D animation facilities take advantage of all the Archimedes screen, colour and plotting modes including dual screens for high speed flicker-free 3D animation - ideal for games & simulations.

Design facilities include 3D Editors to design objects with lines, create symmetrical objects by Sweeping simple sections, use objects as building blocks to create more complex objects which in turn may be used as building blocks, specify individual line colours for multi-coloured objects, dynamic 3D viewing from any angle and an applications facility to animate the 3D models from your own Basic or Machine code programs.

You'll find no better company to do business with than Silicon Vision. As Europe's fastest growing CAD software house, there's no-one more capable of satisfying your future needs. In the meantime we offer you 3D GDS with our compliments. Worth over £25 it can now be yours for only £19.95.



Compatible with Archimedes 305, 310, 410 and 440. Order one today and enter the dynamic world of high speed 3D computer graphics.

SPECIAL OFFER TO RISC-USER MEMBERS ONLY (Quote Membership Number when Ordering).
3D CAD/ANIMATION SYSTEM: £17.95

SILICON VISION

All prices are fully inclusive.
Contact your local dealer or order directly by cheque/P.O./Access/Mastercard/Eurocard from:

SILICON VISION LTD, SIGNAL HOUSE, LYON RD, HARROW, MIDDX, HA1 2AG.
Tel: 01-422 2274 or 01-861 2173. Fax: 01-427 5169. Telex: 918266 SIGNAL G.

ARCHIMEDES SOFTWARE TOOLS

Programmers utility disk No 1 includes:

MicroEMACS: The UNIX industry standard text editor.
Supports Multi-file working, Multi-window, user macros. On-line help.
Access to operating system.
Screen resolution changeable whilst editing

KERMIT: The industry standard intelligent file transfer utility.
Block checking with automatic re-send of faulty blocks (end those RS423 problems!). Multiple file batch transfers, access to the operating system .On-line help facilities.

Plus

UNIX type: HEAD, TAIL, TR, GREP, UUENCODE, UUDECODE, and more

All this for only £38.80 including disc, manuals, VAT and postage

ACCESS / VISA by phone or send cheque to :-

RALPH ALLEN ENGINEERING CO. FORNCETT-END, NORWICH, NORFOLK, NR16 1HT. TEL (BUNWELL) 095389 420

{C} A Dabhand Guide

Learn to program in C on the Archimedes with this latest mammoth Dabhand Guide

A behind the scenes storm has quietly been sweeping over the microcomputer world during the last few years: it is the C programming revolution. So much so that C compilers have been written for all the popular micros including the Archimedes. C looks likely to be the worthy successor to BBC BASIC. On the Archimedes programs written in C are compiled and converted into ARM machine code where they run many times faster.

This latest Dabhand Guide is perhaps the most comprehensive guide to C ever written and covers using Acornsoft C on the Archimedes. In a mammoth 512 pages it introduces the C philosophy in a highly readable, no nonsense manner. Step by step, page by page you can ascend the C ladder. Simple illustrated, documented programs provide the practical type-in theory behind the startling fact.

With thirty-seven chapters, six appendices, glossary and comprehensive index **C: A Dabhand Guide** is amazing value at just £14.95. An Archimedes programs disc is also available for just £9.95 which includes extra programs. Risc User subscribers can obtain both at a special discount.

C: A Dabhand Guide is the *first* book to specifically include coverage of the Archimedes, and is a must for the library of all *serious* Archie users be it at home or work.

Coming next month:

Archimedes Assembly Language: A Dabhand Guide. The first full tutorial on programming the Archimedes in machine code. 368 pages for just £14.95! Watch this space!

How to Order Your Copy

Risc Users may purchase copies of **C: A Dabhand Guide** and programs disc for just £21.95 (normally £24.90) simply by quoting Risc User when ordering. Prices include VAT and P&P. Cheques and POs payable to **Dabs Press** should be sent to the address below. **ACCESS/VISA** accepted by post/phone or mailbox. Add £2 (£10 air) if outside UK.

DABS PRESS (RU) 76 Gardner Rd, Prestwich, Manchester, M25 7HU.
Phone: 061-773-2413 • Microlink 72:MAG11596 • Prestel 94287610

Free Catalogue on Request

DABS PRESS

BACKUP IN BROMLEY

AUTHORISED ACORN DEALERS

We offer full technical backup, service and advice as well as a comprehensive range of software and hardware for your Archimedes. If you have a problem - we can help you.

Call in for a software or hardware demonstration, or phone for prices and availability

Data Store

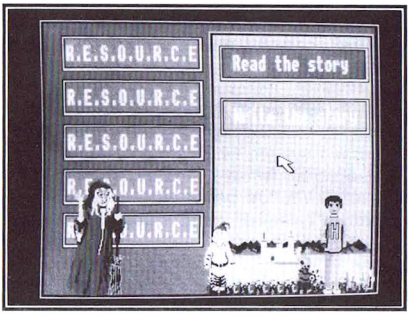
6 Chatterton Road, Bromley, Kent
Telephone : 01-460 8991 (closed Wednesdays)

DESK TOP STORIES

Chris Drage reports on the latest educational software from Resource

Resource is a software publisher with a deservedly excellent reputation in educational computing circles. Having already published Archimedes Droom (reviewed in RISC User Issue 6), Resource has now released Desk Top Stories, an Archimedes version of the popular BBC model B and Master series program Fairy Tales. This review is based on a pre-production version but one which is all but the finished product.

Desk Top Stories provides a simple desk top publishing environment for primary aged children. By combining sprites, graphic characters and text, pages may be created and compiled to make an electronic book. The pages may be then be displayed in sequence, and with the optional print utility disc, dumped to a printer. The package comprises a story disc, sprite library disc and manual.



Resource encourage you to make backups of the story disc in order to create any number of disc-held books (an enlightened attitude). A book can hold up to 40 screen-pages (i.e. one screenful of text, sprites and special character blocks). The library disc contains 126 sprites which can be used to create sprite files with which to illustrate pages. When a new page is selected, one of the sprite-files is assigned to it. Sprites can be selected and moved over the screen at will, and up to 200 sprites may be placed on a page. Replacing and deleting sprites is easily achieved at any time during page composition. Text is presented in double height and in 256 colours. A second set of 26 'picture' characters is available from the keyboard, and these can be combined to depict objects like stone walls, hills etc., large titles or colourful backgrounds. A huge variety of shapes and objects is possible.

User-defined sprites may be created and saved to a file. A user sprite file can contain up to 10 additional sprites. Any sprite can be superimposed on another and by clicking on an icon the former sprite becomes partially obscured by the latter. Thus characters can be made to look over objects. Desk Top Stories utilises the Archimedes WIMP environment to the full, and all control in the program achieved with the mouse. Simple menus are offered with icons being kept to a minimum.

As printing cannot take place from the main program, Resource offers a supplementary disc from which fast, high-quality dumps are obtained. The supplementary disc is the result of a co-operative venture with Clares Micro Supplies in which a special (cut-down), version of the Artisan Support Disc has been adapted to work with Desk Top Stories. A route out of Desk Top Stories is available, and pages must undergo a transformation from mode 13 to mode 12 before printing takes place. Many printer types are supported.

With Desk Top Stories, Resource has managed to achieve a superb balance between sophisticated features and simplicity of operation. The package is immediately comprehensible and easy to manipulate, utilising as it does superb colour graphics and fast speed of operation. Naturally, children love it. In fact, children of all ages will enjoy the experience of creating illustrated stories and poems in such a simple yet powerful environment. In the home, Desk Top Stories will provide your youngster with hours of creative fun and satisfaction (a welcome change from playing endless computer games). But don't be surprised if you become totally engrossed as well!

RU

Title	Desk Top Stories
Supplier	Resource Exeter Road, Off Coventry Grove, Doncaster, DN2 4PY.
Price	£29.95 + VAT (Printer support disc £10.00 + VAT)

3D GRAPHICS DEVELOPMENT SYSTEM

Reviewed by Geoff Bains

The Archimedes is an ideal machine for graphics work so it is good to see some applications taking advantage of this. The 3D Graphics Development System from Silicon Vision is, to be truthful, making little use of the full potential of the micro. This is a re-hashed version of a much earlier package produced for the BBC micro just adapted to work with the 6502 emulator on the Archimedes.

However, the 3D Graphics Development System is still a powerful piece of software which will find many new admirers on the Archimedes. The software is supplied on a single disc which holds both the software and any user data. The package is in two main parts - the model editor and the real-time utilities programs.

The system is exclusively concerned with wire-frame representations of 3D objects. It is not a drawing package so much as a 3D CAD system with animation thrown in. The models are defined in terms of 3D co-ordinates in simple data tables. These are created by typing in the co-ordinates required along with information as to whether the point is to be moved to or drawn to and the colour and type (solid or dotted) of the line. No provision for useful editing of the table is provided. Points can only be deleted and added at the end.

Creating complex shapes in this way calls for some nimble mental gymnastics and so the package provides some additional help in this area. Simple shapes can be rotated about any of the axes to form solids of revolution. Useful solid elements can be treated as 'macros' and added together after suitable scaling, rotation and translation to build up complex models.

Once the data has been entered into the table or created by the software, the model can be viewed on the screen. A perspective projection from any viewpoint can be displayed. The parameters of rotation, scaling, transformation and perspective 'distance' are altered as required with a cursor controlled menu down the side of the screen.

However, the real joy of this software comes after your model has been designed in the model editor. The data table is first 'compiled' and then you can manipulate the model from your own Basic programs. The graphics manipulation routines are incorporated into your own programs so that once written with the supplied software, your finished programs can be run on any Archimedes (with the 6502 emulator).

Several template programs are supplied to serve as starting points for your own work. The Basic program has to load in the model data to one of two specified addresses, and then calls any of a series of procedures (already defined in the template) which actually access the fast machine code routines. In this way a simple Basic program can display a perspective view of a wireframe model from any direction and, with some simple erasing between successive views, complex animation sequences can be created.

As well as 3D co-ordinate manipulation and display routines, the software includes a double screen system which automatically draws on one screen while it is displaying another and then switches between them. This makes simple animation much smoother than can be achieved otherwise.

The 3D Graphics Development System is only a stopgap for Silicon Vision. A fully fledged Archimedes version written in ARM assembler is on the way. However, for a taste of what is possible this package provides a cheap and comprehensive introduction into the world of 3D graphics. **RU**

Product	**3D Graphics Development System**
Supplier	**Silicon Vision** **Signal House, Lyon Road,** **Harrow HA1 2AG.**
Price	**£17.95 inc. VAT** **(Special RISC User price)**

HINTS & TIPS HINTS & TIPS

This month's Hints are rounded up by Lee Calcraft

LABEL-BASED RESTORE

When using more than one group of DATA statements in a program, the normal way to select the required set is to use RESTORE with a line number. Line numbers can easily get altered, they convey little meaning to the programmer, and do not offer the flexibility of labels. The following technique implements a label-based RESTORE.

Begin by defining a procedure *PROCrestore(label$)* as follows:

```
DEFPROCrestore(label$)
RESTORE
REPEAT
READ string$
UNTIL string$=label$
ENDPROC
```

Then just precede each block of data with a label, thus:

```
DATA Manufacturer
DATA Ford,11.4,Saab,27.3,BL,12.6
DATA FordCars
DATA Escort,Sierra,Fiesta,1.6,2.0,1.3
```

Then use PROCrestore("Manufacturer") to "RESTORE" the first block of data, and PROCrestore("FordCars") to "RESTORE" the second, and so on.

This will work well with any number of blocks of DATA, and executes very quickly, even if the procedure needs to check through many DATA items before reaching the required block.

VDU4 AND THE CURSOR

VDU4 is used to cancel the effect of VDU5, which permits printing of text at the graphics cursor. Unfortunately VDU4 re-enables the flashing cursor. This does not occur on earlier BBC micros, and the only answer is to execute an OFF after each VDU4. There also appear to be other problems with cursor blanking which Acorn are looking into.

SCREEN TRANSFER

Because of the different way screens are stored on the model B and Archimedes, it is very hard to transfer from one format to the other. However, an easy way to do the transfer is to use *SPOOL to generate a file containing all the VDU sequences used to draw the screen, then use *EXEC to execute these commands on the other computer. The problem is then reduced to one of transferring a file from the model B to the Archimedes. This can either be done using a serial link, or with a 3.5" ADFS disc on the model B. Thanks to John Ollivere for this tip.

RISC USER DISC MENU ON THE NET

The RISC User Disc Menu (Issues 1 and 2) works perfectly on Econet, and provides an excellent front end for network use. Thanks to Dennis Weaver for this hint.

TWIN HINTS

TWINO8 FOR SPEED

The command TWINO8 will put a program into Twin, and strip off its line numbers. One very good reason for using this option is that Twin returns to Basic many many times faster after this option. With a sizeable program, Twin can take 5 or even 10 seconds to return to Basic, but with TWINO8 it is almost instantaneous.

FINDING YOUR PLACE

The *Find and Replace* option in Twin provides an excellent way of finding your way to a given point in a program. For example, just label a particular part of a program with

```
REM     +++
```

Then press f4 followed by +++ Return, and you are there immediately. You can extend this to finding line numbers if you are not using the TWINO option. Just press f4, enter the line number, press Return, and you are there.

AVOIDING PROGRAM LOSS

When you exit Twin, requesting a return to Basic, the edited file in Twin's buffer is converted into a Basic program and placed at PAGE. To avoid this happening, and to reinstate any program *previously* resident at PAGE, press Return when asked for a language, or enter **N** Return if asked to confirm a return to Basic. This will return you to Arthur. Then type:

```
BASIC
OLD
```

and you will have your old program back. This technique can be very useful when using Twin to edit a data file for testing with a program resident at PAGE. Thanks to Nic van Someren for this tip.

RISC USER magazine

MEMBERSHIP

RISC User is available only on subscription at the rates shown below. Full subscribers to RISC User may also take out a reduced rate subscription to BEEBUG (the magazine for the BBC micro and Master series).

All subscriptions, including overseas, should be in pounds sterling. We will also accept payment by Connect, Access and Visa, and official UK orders are welcome.

RISC USER SUBSCRIPTION RATES

		RISC USER & BEEBUG
£14.50	1 year (10 issues) UK, BFPO, Ch.I	£23.00
£20.00	Rest of Europe & Eire	£33.00
£25.00	Middle East	£40.00
£27.00	Americas & Africa	£44.00
£29.00	Elsewhere	£48.00

BACK ISSUES

We intend to maintain stocks of back issues New subscribers can therefore obtain earlier copies to provide a complete set from Vol.1 Issue 1. Back issues cost £1.20 each. You should also include postage as shown:

Destination	UK, BFPO, Ch.Is	Europe plus Eire	Elsewhere
First Issue	40p	75p	£2
Each subsequent Issue	20p	45p	85p

MAGAZINE DISC

The programs from each issue of RISC User are available on a monthly 3.5" disc. This will be available to order, or you may take out a subscription to ensure that the disc arrives at the same time as the magazine. The first issue (with six programs and animated graphics demo) is at the special low price of £3.75. The disc for each issue contains all the programs from the magazine, together with a number of additional items by way of demonstration, all at the standard rate of £4.75.

MAGAZINE DISC PRICES	UK	Overseas
Single issue discs	£ 4.75	£ 4.75
Six months subscription	£25.50	£30.00
Twelve months subscription	£50.00	£56.00

Disc subscriptions include postage, but you should add 50p per disc for individual orders.

All orders, subscriptions and other correspondence should be addressed to:

RISC User, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX.
Telephone: St Albans (0727) 40303
(24hrs answerphone service for payment by Connect, Access or Visa card)